CMP
United Business Media

PRODUCT REVIEWS: INTEL'S VTUNE 6, BIONATICS' NATFX, MORE

Game Developer

AUGUST 2002

ON THE FRONT LINE OF GAME INNOVATI

Scatter Logic

Outdoor Light Scattering Techniques

More Bounce For Your Buck

Integrating a Physics Engine

Postmortem

Treyarch's SPIDER-MAN

WWW.GDMAG.COM

$5.95US $6.95CAN

08

0 71486 02133 9

Display Until August 23, 2002

Sends other UNIX

"My PowerBook G4 is now running every major UNIX app that we had on our Suns, AlphaStations and SGIs – and running them faster."

– Mark S. Cohen, Ph.D., Professor, UCLA Brain Mapping Center

"This new OS has accomplished in a short period of time what others have struggled to do for years: bring a compelling, widely accepted GUI (called Aqua) to UNIX."

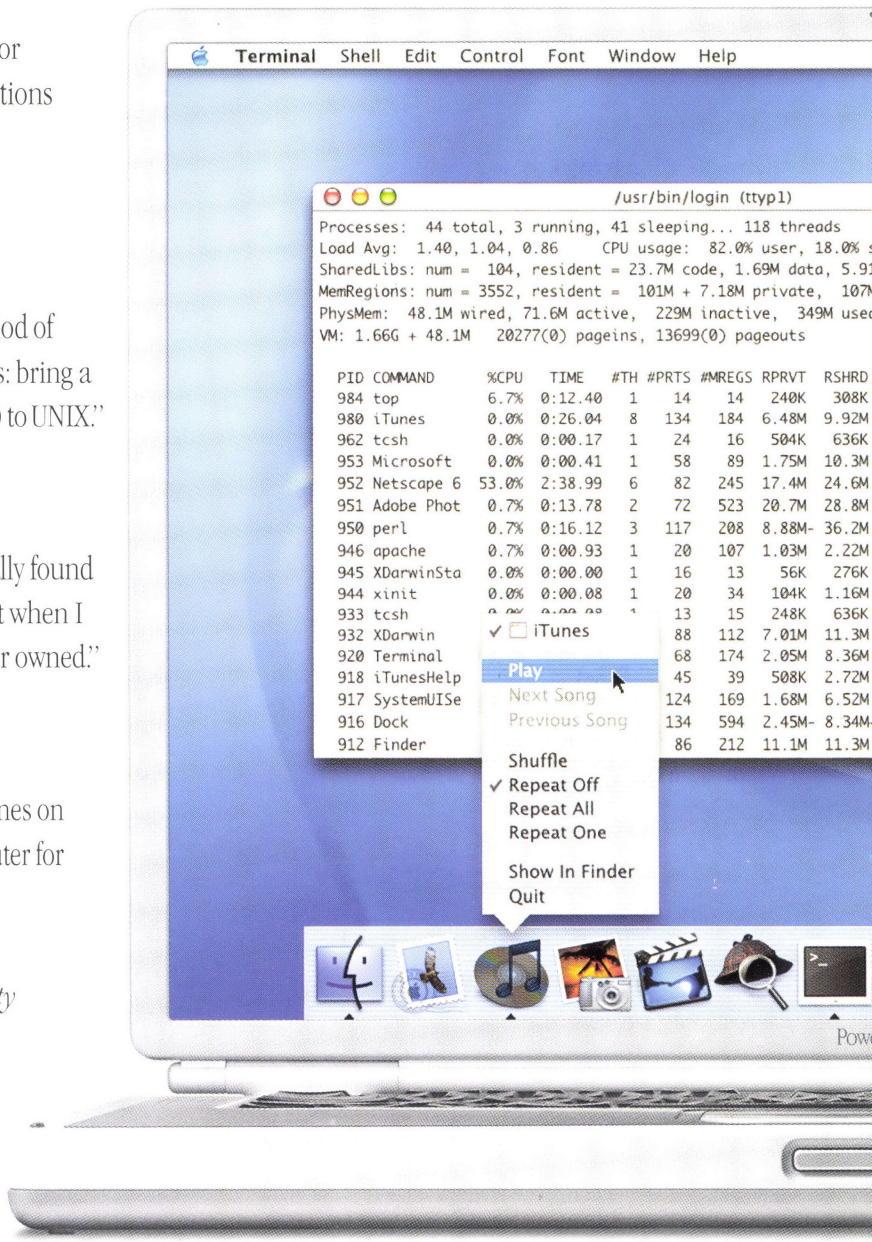
– Tim O'Reilly, O'Reilly Network

"After two-and-a-half years of Linux, I've finally found joy in a UNIX operating system. And I found it when I purchased a Macintosh – the first one I've ever owned."

– John Hummel, Jr., The Gamers' Press

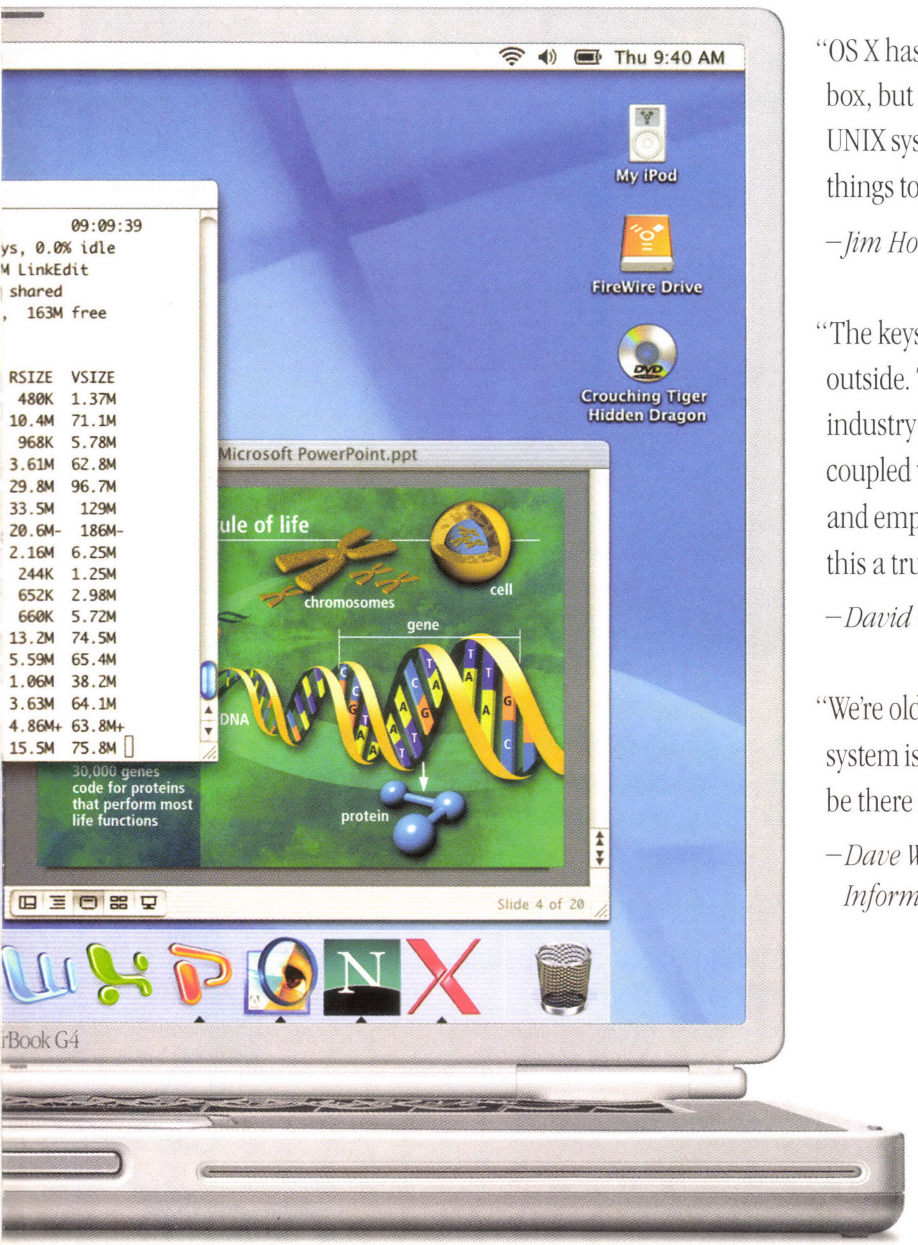
"Until Mac OS X came along, I had three machines on my desk. Now with Mac OS X, I use one computer for writing, coding, research, everything."

– Dr. Michael Cherry, Associate Professor, Department of Genetics, Stanford University



TM and © 2002 Apple Computer, Inc. All rights reserved. Some software titles not included.

boxes to /dev/null.



"OS X has all the power you expect from a UNIX box, but none of the hassle associated with basic UNIX system administration. You can expect things to work and they do."

—Jim Hourihan, Tweak Films

"The keys to OS X are that it's UNIX inside and Mac outside. This combination of stability, support for industry standards, and wide developer support coupled with Apple's Aqua graphical user interface and emphasis on graphic performance makes this a truly great operating system."

—David Coursey, ZDNet AnchorDesk

"We're old hardcore UNIX hackers, so a BSD-based system is mother's milk. Everything you expect to be there is there, and it works right."

—Dave Weininger, President, Daylight Chemical Information Systems, Inc.

For more information, call 1-800-MY-APPLE or visit www.apple.com/macosx.

Buy an engine. Gain a partner.

When you license NDL's NetImmerse 3D game engine and development toolkit, you get much more than great code. You gain a development partner with engineers dedicated to doing everything possible to ensure the success of your title.

NetImmerse is the only engine that can deliver all of this:

- A complete toolset that accelerates your art pipeline.
- Optimized performance across all platforms.
- Full source code for all libraries and tools.
- Responsive support from those who develop the engine.

Great software. Solid partner. Proven success.

NetImmerse

The fastest path from conception to completion

925.556.9995 • sales@ndl.com

ndl

The Elder Scrolls III: Morrowind courtesy of Bethesda Softworks.
Star Trek® Bridge Commander™ courtesy Paramount Pictures. ™, ®, & © 2002 Par. Pic. All Rights Reserved.
Dark Age of Camelot courtesy of Mythic Games.
Tennis Master Series courtesy of Microids.
Freedom Force courtesy of Irrational Games.

www.ndl.com

CONTENTS

VOLUME 9, NUMBER 8

August 2002

FEATURES

32 Photorealistic Real-Time Outdoor Light Scattering
The secrets of the sun revealed to you! Well, not really, but Naty Hoffman and Arcot Preetham lay down the groundwork to help you build a scattering model and implement it using a vertex shader.
Naty Hoffman & Arcot J. Preetham

40 Outsourcing Reality: Integrating a Commercial Physics Engine
As game developers reach for new forms of gameplay and a better process for implementing established genres, the wisdom of licensing physics technology is becoming inescapable.
Matt MacLaurin

48 Treyarch's Spider-Man
In bringing the adventures of our favorite web swinger and slinger from the silver screen to gameplayers' TV screens, James Fristrom and the rest of the crew navigated through a web of multi-console challenges, deadlines, and heroics of their own.
James Fristrom

DEPARTMENTS

4 **GAME PLAN** *Jennifer Olsen*
Growing up, Not Growing Old

6 **SAYS YOU**
Letters from our readers

8 **INDUSTRY WATCH** *Daniel Huebner*
Interplay unravels, Nintendo profits, Xbox heads online, more

10 **PRODUCT REVIEWS**
Intel's VTune, Bionatics NFX, and Geodesic's Great Circle 6.0.0.9

16 **PROFILES** *Jennifer Olsen*
VIS Entertainment's Chris van der kuyl

COLUMNS

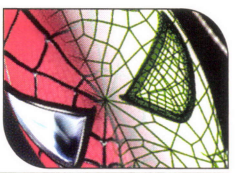
18 **THE INNER PRODUCT** *Jonathan Blow*
Rendering Level-of-Detail Forecast

22 **ARTIST'S VIEW** *Hayden Duvall*
Building the Future, Part 2: Vehicles

28 **SOUND PRINCIPLES** *Aaron Marks*
Blending the Total Soundscape

30 **BETTER BY DESIGN** *Noah Falstein*
Asymmetrical Distribution

64 **SOAPBOX** *Paul Warne*
The Games of Art



COVER *composited from images provided by Treyarch.*

Game Developer (ISSN 1073-922X) is published monthly by CMP Media LLC, 600 Harrison St., San Francisco, CA 94107, (415) 947-6000. Please direct advertising and editorial inquiries to this address. Canadian GST: 124513185. **Subscription Rates:** Subscription rate for the U.S. is $49.95 for twelve issues. Countries outside the U.S. must be prepaid in U.S. funds drawn on a U.S. bank or via credit card. Canada/Mexico: $69.95; all other countries: $99.95 (issues shipped via air delivery). Periodical postage paid at San Francisco, CA and additional mailing offices. **Postmaster:** Send address changes to Game Developer, P.O. Box 1274, Skokie, IL 60076-8274. **Customer Service:** For subscription orders and changes of address, call toll-free in the U.S. (800) 250-2429 or fax (847) 647-5972. All other countries call (1) (847) 647-5928 or fax (1) (847) 647-5972. Send payments to *Game Developer*, P.O. Box 1274, Skokie, IL 60076-8274. For back issues write to *Game Developer*, 1601 W. 23rd St. Ste. 200, Lawrence, KS 66046-2703. Call toll-free in the U.S./Canada (800) 444-4881 or fax (785) 841-2624. All other countries call (1) (785) 841-1631 or fax (1) (785) 841-2624. Please remember to indicate *Game Developer* on any correspondence.

GAME PLAN

LETTER FROM THE EDITOR

Growing up, Not Growing Old

"So, what do you think of the show?" This question is the most basic opening gambit of tradeshow banter. After a couple of days (or sometimes only hours) spent fighting the sensory assault of game industry events, sometimes its pure simplicity holds your only hope of stringing together an intelligible thought to communicate to another person. Whether engaged in this exchange myself or merely observing another conversation, I lost count of the number of times this year at GDC and E3 that people's response to this question was to gaze wistfully off into the distance and say something to the effect of, "It all seems so . . . *grown up*."

No one seemed quite sure whether that was a good thing or a bad thing, but there is a measure of truth to the sentiment. The tangible business aspects of industry events such as E3 have come more clearly into the fore, making welcome inroads against the seedy sideshow angle. I didn't bring my tape measure or calipers to E3 this year, so I lack empirical data, but even the smarmy booth babe presence seemed to be on the decline. There was less noise for the sake of noise, less latex for the sake of latex. Not none, but less.

It makes sense; there's more real business to do every year as the industry grows. But what I also see is the game industry beginning to feel truly comfortable in its own skin. We're no longer the brooding stepchildren of other entertainment or technology sectors but confident that our unique offerings can hold their own in the mainstream, and do so on our terms, not as an adjunct to another industry chafing at our continued presence. And while there are still a small token handful of Hollywood-wannabes out there embarrassing themselves on our behalf, many in the industry feel within striking distance of finding the path to Hollywood-level success without ending up in a gutter on the Sunset Strip.

We're moving beyond the days of the developer with the rock-star complex striking off to run his own studio, only to find out that running a business is more than putting a ROBOTRON machine in the breakroom and giving employees free soda. Development professionals eyeing their future don't want to be roadies on tour with a rock star, they want a solidly run company that stands a good chance of succeeding and producing quality products. With publishers' tolerance of risk nearing all-time lows, a consistent track record of sensible business dealings is any development studio's best asset right now.

But as our success and growth outlook lead to more attention on the serious business of game development, are we losing part of our identity by showing a more grown-up face? When game development transitioned from solo programmers working a project to teams of programmers, artists, and designers, some developers couldn't adapt and left the industry for good, taking their talent and experience with them. Does "growing up" mean alienating another segment of rare talent who were first drawn to game development because of the eccentric, laid-back, and decidedly unpretentious environment?

The trick will be to continue to focus on what has brought the industry success in the first place: offering consumers a kind of entertainment experience they can't get from any other medium. As long as we continue to nurture and develop the unique interactive essence of games while continuing to improve on the overall production values, quality of entertainment, and variety of our offerings, we may yet be able to retain the identity that got us this far.

Jennifer

Jennifer Olsen
Editor-In-Chief

GameDeveloper

600 Harrison Street, San Francisco, CA 94107 t: 415.947.6000 f: 415.947.6090

Publisher
Jennifer Pahlka *jpahlka@cmp.com*

EDITORIAL

Editor-In-Chief
Jennifer Olsen *jolsen@cmp.com*

Managing Editor
Everard Strong *estrong@cmp.com*

Production Editor
Olga Zundel *ozundel@cmp.com*

Product Review Editor
Daniel Huebner *dan@gamasutra.com*

Art Director
Elizabeth von Büdingen *evonbudingen@cmp.com*

Editor-At-Large
Chris Hecker *checker@d6.com*

Contributing Editors
Jonathan Blow *jon@bolt-action.com*
Hayden Duvall *hayden@confounding-factor.com*
Noah Falstein *noah@theinspiracy.com*

Advisory Board
Hal Barwood LucasArts
Ellen Guon Beeman Beemania
Andy Gavin Naughty Dog
Joby Otero Luxoflux
Dave Pottinger Ensemble Studios
George Sanger Big Fat Inc.
Harvey Smith Ion Storm
Paul Steed WildTangent

ADVERTISING SALES

Director of Sales/Associate Publisher
Michele Sweeney e: *msweeney@cmp.com* t: 415.947.6217

Senior Account Manager, Eastern Region & Europe
Afton Thatcher e: *athatcher@cmp.com* t: 415.947.6224

Account Manager, Northern California & Southeast
Susan Kirby e: *skirby@cmp.com* t: 415.947.6226

Account Manager, Recruitment
Raelene Maiben e: *rmaiben@cmp.com* t: 415.947.6225

Account Manager, Western Region & Asia
Craig Perreault e: *cperreault@cmp.com* t: 415.947.6223

Account Representative
Aaron Murawski e: *amurawski@cmp.com* t: 415.947.6227

ADVERTISING PRODUCTION

Vice President, Manufacturing Bill Amstutz
Advertising Production Coordinator Kevin Chanel
Reprints Cindy Zauss t: 909.698.1780

GAMA NETWORK MARKETING

Director of Marketing Greg Kerwin
Senior MarCom Manager Jennifer McLean
Marketing Coordinator Scott Lyon

CIRCULATION

BPA INTERNATIONAL
Game Developer is BPA approved

Group Circulation Director Catherine Flynn
Circulation Manager Ron Escobar
Circulation Assistant Ian Hay
Newsstand Analyst Pam Santoro

SUBSCRIPTION SERVICES

For information, order questions, and address changes
t: 800.250.2429 or 847.647.5928 f: 847.647.5972
e: *gamedeveloper@halldata.com*

INTERNATIONAL LICENSING INFORMATION

Mario Salinas
t: 650.513.4234 f: 650.513.4482 e: *msalinas@cmp.com*

CMP MEDIA MANAGEMENT

President & CEO Gary Marshall
Executive Vice President & CFO John Day
Chief Operating Officer Steve Weitzner
Chief Information Officer Mike Mikos
President, Technology Solutions Group Robert Faletra
President, Business Technology Group Adam K. Marder
President, Healthcare Group Vicki Masseria
President, Electronics Group Jeff Patterson
President, Specialized Technologies Group Regina Starr Ridley
Senior Vice President, Global Sales & Marketing Bill Howard
Senior Vice President, HR & Communications Leah Landro
Vice President & General Counsel Sandra Grayson
Vice President, Creative Technologies Philip Chapnick

CMP United Business Media

GamaNetwork

WWW.GAMANETWORK.COM

FORGET EVERYTHING YOU'VE HEARD.

SOUNDMAX SPX

Someday, you'll look back and remember the moment sound changed.

Enter SoundMAX SPX, powered by the SoundMAX Audio Rendering Technology Toolkit (SMartTools). Create ultimate, immersive audio, using advanced audio algorithms to deliver hyper-realistic game experiences.

We've blown through the sound barrier. And warped into the next generation of sound dynamics. Now you can sculpt, manipulate and control complex audio behaviors. Form real-time sound events that you animate on-the-fly as the game is played.

Exploit the power of SoundMAX in your next project. Fully scalable. Cross platform. It's your choice. Get SMart. Sound better.

Remember where you heard it first.

WWW.AUDIOFORGAMES.COM

ANALOG DEVICES

SoundMAX® is a registered trademark of Analog Devices, Inc.

SAYS YOU

A FORUM FOR YOUR POINT OF VIEW. GIVE US YOUR FEEDBACK...

Safety First

Scripting has become quite commonplace in game development, and the questions of which language to use plague every project design. In "Creating a C++ Scripting System" (June 2002), Emil Dotchevski makes many references to the safety of the scripting language and finding ways to prevent users from crashing the game by invalid memory access (bogus pointers).

While I agree this is a big issue for scripts, there are much more significant safety issues (such as access to critical system functions) you must consider when using any compiled language that will be loaded by the system. In fairness, the titles listed in Mr. Dotchevski's credits are console titles and scripting is far more controlled. For PC titles, we aren't so lucky.

We look bad when a newly downloaded mod crashes our game, but we look like total idiots when that mod crashes the system by modifying the registry, fills our hard drives with bogus files, modifies our work, or uploads our information to a remote server without our consent.

Script safety needs extreme scrutiny. Unfortunately, safety issues of this nature are all too frequently an afterthought, or not a thought at all, when considering our scripting language. C++ certainly is viable as a scripting language, but the way it is packaged and loaded requires a good deal of care.

Cliff Owen
Goblin Software

XSI Animation Mixer Not That Recent

Steve Theodore's article "Understanding Animation Blending"(May 2002) was clear, informative, and timely. I wanted to comment on the reference to XSI's animation mixer, described as "most recent and perhaps the most powerful" among a list of similar technologies. It is far from the most recent. In fact, we were the first of the systems listed in the article to commercialize timeline-based animation mixing. It was a core feature of XSI 1.0 way back in May 2000, predating similar functionality found in Maya, Max, and Lightwave.

Michael Sheasby
Softimage

With the widespread availability of graphics hardware possessing per-pixel lighting capabilities, the days of applying only one kind of texture to each surface are numbered.

Smoothing Over Bumps

I was both pleased and disappointed when I read Hayden Duvall's Artist's View column "From Source to Texture" in the June 2002 issue. As a 3D graphics engine programmer, I have a real appreciation for Hayden's advice that artists retain versions of their work at the highest possible level of detail. In the past, I have seen situations in which a development team has kicked themselves for not saving a higher-resolution or higher-color-depth version of some textures created a year earlier.

I wish the article had talked more about designing textures to be used with modern per-pixel lighting engines. In several places, the article mentions the addition of lighting and shadows to texture maps, but I think artists need to get used to not doing this because modern graphics hardware now enables this information to be calculated in real time. Hayden says that "in-game lighting can often have a flattening effect" on textures, but the opposite statement is true when bump maps are used. For a bump-mapped object to be rendered correctly, its texture should contain absolutely no lighting or shadowing information — you actually want it to be flat. The texture shown in Figure 2 would perform particularly badly in a per-pixel lighting engine due to its pre-lit appearance. The texture shown in Figure 6 would probably look all right when used with a bump map, but it would be better if the shadows were removed and the specular highlights flattened out.

With the widespread availability of graphics hardware possessing per-pixel lighting capabilities, the days of applying only one kind of texture to each surface are numbered. Artists will have to learn to separate the surface shading information that used to be combined in a single texture and adapt to the process of creating a base texture containing only flat color information, a bump map containing depth information, a gloss map containing reflectance information, and the list continues.

Eric Lengyel
Terathon Software

CORRECTION

Jennifer Olsen's July 2002 "Game Plan" column ("So It's Come to This") incorrectly referenced H.R. 4652 as the Protect Children from Video Game Sex and Violence Act. The correct bill number is H.R. 4645. H.R. 4652 is the Consumer Protection for On-Line Games Act, which was introduced in Congress the same day. We regret the error.

E-mail us your feedback to editors@gdmag.com, or write us at *Game Developer*, 600 Harrison St., San Francisco, CA 94107

FACIAL ANIMATION STRESSING YOU OUT?

You Need Eyematic® FaceStation™

The World's First Intelligent Facial Animation System™

Got a briefcase full of money for proprietary motion capture hardware and software? Got a team of highly-skilled motion capture engineers? Got an army of animators to tweak keyframes? Got any aspirin?

Eyematic puts an end to the madness - with FaceStation, you no longer have to be rich or famous to create high-quality facial animation. Eyematic's patented computer vision software automatically locates and tracks facial expressions and mouth movements in real-time, using just a single video camera or camcorder. No awkward facial markers, no time consuming post-processing and no expensive dedicated hardware. Just an integrated suite of software applications that lets anyone easily create expressive, realistic facial animation.

Visit our web site to find out how FaceStation can add the power of professional facial animation to your tool chest.

www.eyematic.com/face

Eyematic FaceStation
Only U.S. **$1,995**

Version 1.5 Now Shipping for Discreet™ 3ds max™ Users
Alias | Wavefront® Maya® Version Available Summer 2002

EYEMATIC®

Copyright © 2002 Eyematic Interfaces, Inc. Pricing is suggested retail pricing only, and may also vary by country. Eyematic is a registered trademark and FaceStation and Intelligent Facial Animation System are trademarks of Eyematic Interfaces, Inc., but you could probably guess that. Other product or service names mentioned herein are the trademarks of their respective owners.

INDUSTRY WATCH

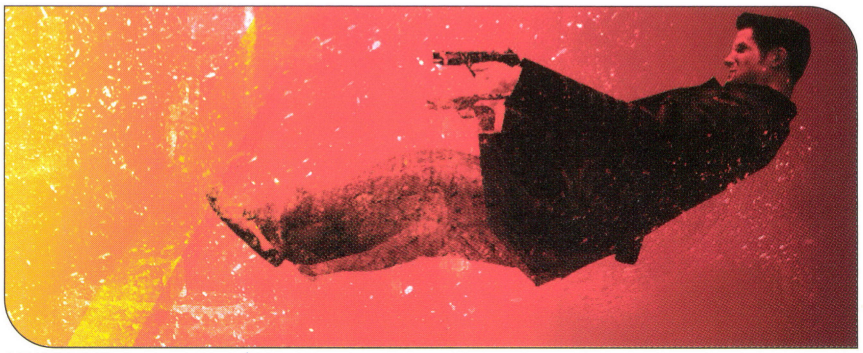
THE BUZZ ABOUT THE GAME BIZ | daniel huebner

Interplay unravels. Despite posting its first quarterly profit in nearly two years, Interplay's financial self-destruction shows no sign of abating. The company was able to post net income of $1.5 million for the first quarter of 2002, a sizeable improvement from the $8.4 million loss in the same period one year ago.

The positive results may not be enough to rescue the debt-ridden company, however. Interplay's most recent filing with the Securities and Exchange Commission reports $54 million in debts and just $61,000 in cash on hand. Even with the $47 million the company brought in through the sale of Shiny to Infogrames, Interplay admits that it needs to raise substantial amounts of new financing to stay afloat. In an effort to raise capital, Interplay recently sold the rights to many of its game properties — including MESSIAH, MDK, KINGPIN, SACRIFICE, and EARTHWORM JIM — to Titus Interactive.

Nintendo reaches record profits, Yamauchi takes a bow. A weak yen couple with strong Gamecube sales equaled record profit for Nintendo in its fiscal year 2001. The company posted a net profit of 106.4 billion yen ($858 million), over 10 percent higher than Nintendo's profits last year. While the company cited Gamecube and Game Boy Advance sales as its growth leaders, the yen paints a big part of Nintendo's financial picture, and a strengthening Yen has prompted Nintendo to warn that fiscal 2002 likely won't reach the lofty heights the company enjoyed this year.

The strong financial results offered a perfect opportunity for Nintendo's chairman and president, Hiroshi Yamauchi. Yamauchi to announce that he is stepping down after leading the company for more than five decades. Yamauchi's retirement was long expected, and the news did not affect Nintendo's share price. He will be kept on as an advisor but will not be part of the company's management team. Satoru Iwata, the company's chief of corporate planning, has been appointed Yamauchi's successor.



MAX PAYNE. A man with a $34 million price tag on his head.

Prior to joining Nintendo in 2000, Iwata was president of HAL Laboratory, where he coordinated software development and production of several Nintendo games, including the KIRBY series.

Secure no more, Xbox heads online. A graduate student at the Massachusetts Institute of Technology discovered a way to foil the Xbox's security system, in theory making it possible to run non-Xbox applications on the console. In a paper posted on the MIT web site, Andrew Huang, a Ph.D. student, describes a "secret boot block" in the Xbox's media processor (built by Nvidia) which contains the console's encryption algorithm and security key. The boot block was decrypted using a custom "tap board," which intercepted data traveling between the central processor and the media chip over the Hyper Transport bus.

Separate from any security concerns, Microsoft has announced its online strategy for the Xbox. Directly opposite to the hands-off online plans laid out by Sony and Nintendo, Microsoft plans to invest much of its $2 billion Xbox development budget for the next five years into a proprietary subscription-based online service called Xbox Live. The Xbox Live service will debut in Europe, North America, and Japan this fall at the tentative price of $49.95 per year and will let people play online and download content to the Xbox hard drive. Microsoft expects to have over 50 online games available for the Xbox by the end of 2003, including STAR WARS GALAXIES, HALO, and PROJECT GOTHAM RACING. Microsoft will build new data centers in Seattle and Tukwila, Wash., Tokyo, and London to support the service.

No Payne, no gain. MAX PAYNE is moving from Remedy Entertainment to Take-Two for $34 million in cash and stock. The deal includes all intellectual property rights associated with the brand, including trademarks, copyrights, characters, perpetual license to utilize proprietary technologies (including the MAX PAYNE game engine), and rights to license fees from ancillary MAX PAYNE brand extensions such as cinema, television, and literary productions. A MAX PAYNE sequel is in the works from Take-Two's Rockstar Games in conjunction with original MAX PAYNE developer Remedy Entertainment.

UPCOMING EVENTS CALENDAR

GDC EUROPE

EARL'S COURT
London, U.K.
August 27–29, 2002
Cost: £350–£450
www.gdc-europe.com

ECTS

EARL'S COURT
London, U.K.
August 29–31, 2002
Cost: Advance registration free via web site; £25 on-site registration
www.ects.com

3ds max

Every day, gamers buy more games created with **3ds max**™ than any other animation software. Which makes sense when you consider **3ds max** has helped create thousands of games in every genre and platform. Whether it's RPG, Strategy, Simulation or good clean fun plundering Glukkons, all **3ds max** games have one thing in common: Gamers can't seem to get enough.

discreet™

For more info on **3ds max** and other 3D products like **reactor**™ and **character studio**®, call 1.800.869.3504 or go to www.discreet.com.

Discreet is a division of Autodesk, Inc. Discreet, character studio, 3ds max, and reactor are either registered trademarks or trademarks of Autodesk, Inc. in the U.S.A. and/or other countries. All other brand names, product names or trademarks belong to their respective holders. ©2002 Autodesk, Inc. All rights reserved. ©1987, 2001 Konami Computer Entertainment Japan, Planet Moon Studios, Oddworld: Munch's Oddysee ©2002 Oddworld Inhabitants, Inc. All Rights Reserved. The Oddworld logo, Oddworld: Munch's Oddysee are either registered trademarks or trademarks of Oddworld Inhabitants in the United States and/or other countries. Lionhead Studios/Electronic Arts™, Diablo® II box cover image courtesy of Blizzard Entertainment®, Remedy Entertainment & 3D Realms Entertainment, ©1999-2002 Activision, Inc. and its affiliates. Activision is a registered trademark and Activision 02, Tony Hawk's Pro Skater and Pro Skater are trademarks of Activision, Inc. and its affiliates. All Rights Reserved. Tony Hawk is a trademark of Tony Hawk. The ratings icon is a trademark of the Interactive Digital Software Association. Image courtesy of Electronic Arts™, 2002, ©Microsoft Corporation. All rights reserved. Microsoft, Bungie, Halo, Xbox and the Xbox logo are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Valve Software/Sierra.

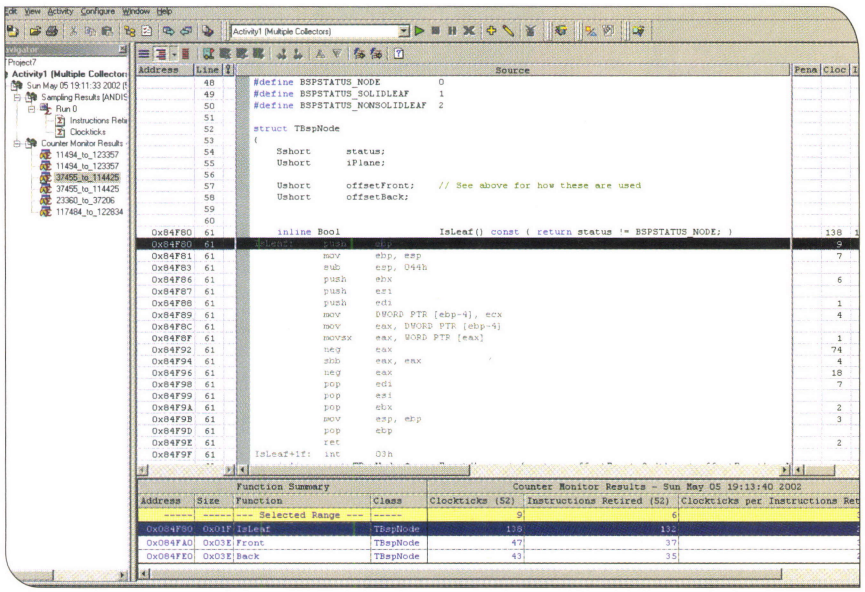
Intel's VTune 6 Performance Analyzer

by andi smithers

Intel's VTune has a long-standing reputation as one of the better tools for application analysis — at least for applications headed for Intel-based systems. I hadn't touched it since version 3.5, and I was more than curious to see what new improvements were implemented in the new version 6. I wasn't disappointed either.

I had actually used the Xbox version of VTune only a few months earlier, and the install process had been more than a little painful, so I was a bit nervous. However, apart from a few redundant reboots and a small problem recognizing the installed version of Flash, everything went smoothly.

I created a quick wizard project, turned on call graphing, and off it ran. After 20 seconds, the project stopped, reran, and then did it again. Was this a bug? On reading the tutorial further, I discovered that the first pass was a calibration pass, the second pass was the actual samples, and the final pass was call graphing. Simple enough, but why did it all only take 20 seconds? It seems that is the default execution time; this might not work well if that time is spent loading, though fortunately you can modify activities and increase this time. Even without call graphing you'll need the calibration pass, so I manually aborted that after a minute or two, and then left the main sampling pass for a few minutes. There is also a simple option for the project to run an application with sampling paused and then have the user manually resume, so you can add your own hooks in your code to generate resume/pause messages using a vtuneperformance.dll. This came in very handy for isolating samples to specific areas (for example, if the frame rate drops, call a resume function to start logging samples).



Intel's VTune Performance Analyzer in action.

I made numerous attempts with sampling, trying to get a good representation, and it didn't take long for me to remove the call graphing. I think this is a feature to turn on only in trouble spots, as it's very intrusive in execution timing and slows down a game to the point where it's not really useful unless you know what you're looking for — or if you're smart and have set up some prerecorded joystick presses that can walk through a game perfectly every time (I'm not that smart).

The results I got without the call graph were great starting points. I recorded about five minutes of samples and counter events. After sampling, the data is displayed in graph form representing everything from CPU percent time, privileged CPU percent, page misses, thread queuing, and more. Intel has supplied a lot of mechanisms to view this data, including graphing as splines, blocks, solid, or wire form, and it's all very customizable. I chose a spline form, though I recommend playing with the display a bit as it does impact how you

ANDI SMITHERS | *Andi is lead engineer at The Collective and currently resides in Newport Beach, Calif. You can reach him at andis@collectivestudios.com.*

perceive the data execution. Using another icon, I selected a time range in order to investigate a peculiar spike in processor-privileged time that looked odd. I highlighted a small, one-second range, hit the drill-down icon, and was rewarded with a much more detailed breakdown. At this point I had numerous modules (DLLs and also the EXE) I wished to look at, but I couldn't merge all the modules together. It's a minor annoyance, but one I can live with until version 7.

VTUNE 6 ★★★★

STATS

INTEL
Santa Clara, Calif.
(408) 765-8080
www.intel.com

PRICE
$699 (MSRP)

SYSTEM REQUIREMENTS
Hardware: At least an Intel Pentium III processor-based system and 128MB of RAM. An Intel Celeron, Intel Pentium II, Pentium II Xeon, Pentium III, Pentium III Xeon, Pentium 4, or Intel Xeon, or Intel Itanium processor-based system for event-based sampling. Note: Event-based sampling is not supported on mobile processors. Software: Microsoft Windows 98 (SE), Windows ME, Windows NT 4.0 with Service Pack 4 or later, Windows 2000 Build 2195 or later, or Windows XP Build 2475 or later. Microsoft Internet Explorer 5.0 or later (5.5 or newer recommended).

PROS

1. Excellent tutorial.
2. Great VTune assistant.
3. Easy to get into and get real info out of.

CONS

1. Could have better integration back into Visual Studio for editing source.
2. Windows could be better sized and positioned, as it gets a little cluttered.
3. Calibration phase was frustrating.

Now, it was a simple case of double clicking on the desired module to bring up a detailed source breakdown, though it did ask me to specify the DSP (project or makefile). This led to my second problem: the project I was debugging has numerous DLLs as well as the main EXE, but the system only wanted to accept a single project file. I wanted it to ask for my workspace from Visual C++ (the .DSW), as it is an extremely complex source base. The good news for Java and other non-C++ people is that VTune does allow you to specify a multitude of project types. I was very happy to see Java, .NET, and even FORTRAN supported (does anyone really still use that?).

By now I've screamed a few times in my head, and once out loud, when my eyes gravitate to the top 10 functions that stall out the execution. Intel's terms are CPU Clockticks (non-sleep) and Instructions Retired (sounds like a CIA euphemism for a shagged compiler). One particular surprise VTune found was a bit of code that looked harmless enough, but when I viewed the source with disassembly it showed the NEG assembler instruction was kicking the function's teeth in, taking a bite out of its performance, and doing this twice. After a quick fix, it dropped off the top-10 list completely.

I tried another of my top-10 items, and rather than trying to find a solution myself, I just right-clicked on the source and selected "VTune Assistant, This Function" (you can also do this to a selection or an entire file). Now this is where I started to get impressed. The assistant returned about 20 occurrences of problems with nice little light bulbs at the lines concerned. It also offered a light bulb at the end of the function, with suggestions on a general problem and solution. What I was very pleased to see was the comment "Logical AND/OR statement conditional," which offered me a very informative description of what it believed could be done.

VTune's assistant is one I would gladly hire, or at least get writing PS2 code. It also caught the loop invariant catch, where you resolve a pointer to a pointer within a loop or a for statement that has the count of a pointer to a class — nasty stuff. But my favorite feature and the one I am very interested to do more with is vectorization, which crops up a lot with virtuals and templates. If you're a progressive C++ engineer who likes to template array handling, then VTune's going to ring your bell, because one of its recommended optimizations is to recommend the new Intel C/C++ compiler version with the new supported vectorization pass, meaning that the compiler will use SSE instructions for some loop operations. I saw this compiler at GDC this year, and it was very impressive (at least by this feature). Beyond that, it recommends restructuring the code to allow for better vectorizing.

I'm generally very happy with the assistant, though I thought it lacked a single critical feature, which really got to be frustrating. The assistant is in its own window, and it displays the line number, but I can't click on the line number and have the window scroll to the correct line of source, so I had to scroll down manually to the line number. This became annoying, especially when I started to do class analysis over function analysis and wanted large blocks of code to be analyzed by the assistant so that I could just scroll through the trivial fixes. Another problem was I could not jump to code in the editor. For those who have used SN Systems' debugger, it has a hot key, Ctrl+E, that jumps to code in Visual C++. I sorely missed this when VTuning my data.

All in all, I thought VTune 6 was extremely good, very stable, and the tutorial was insightful and valuable, making my life significantly easier. The in-depth help, and the fact that hitting F1 on any item brought up the correct context help menu, was invaluable. I had only one crash during a marathon six-hour session. Even when I did crash, when I got back up and running, my project was intact, and it was just a minor inconvenience. VTune 6 is a must for developers, and you don't need to be an assembler wiz to use it (though it does help).

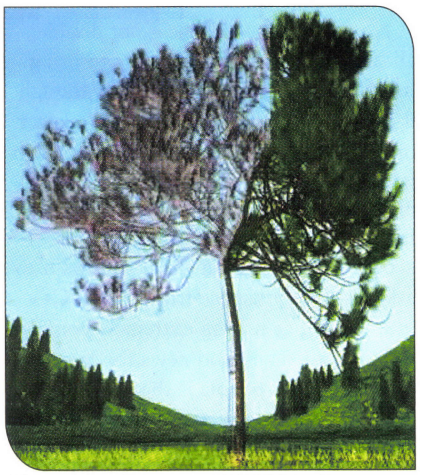
BIONATICS' NATFX

by daniel sánchez-crespo

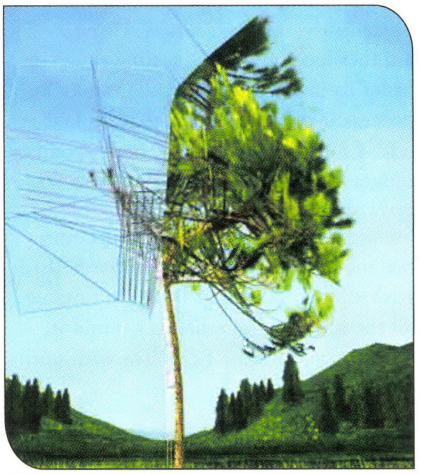
Graphics hardware is evolving at a gigantic pace. Gone are the days of low-polygon meshes; today's games allow rendering of very complex geometry and are opening the doors to representing life-like, lush, organic environments. One of the elements that has only recently found its way to desktop graphics are realistic trees, light years away from the classical billboards found in games of yesteryear. Still, tree modeling can be a complex art: the developer must cleverly find a balance between real geometry and textured planes, often with the help of some level-of-detail processing. This makes tree modeling a time-consuming task, and results are often unpredictable. To ease this task, a number of specific tree-modeling tools have appeared. One of these is Bionatics' NatFX, designed specifically with game developers in mind.

NatFX, a Max plug-in (a Maya version is also available) that allows the rapid creation of trees and plants of various types, is the result of almost 20 years of research dealing with synthetic plant representation, growth, and visualization, carried out at the CIRAD institute in France. Essentially, the package can create all types of plants, which are encoded in a virtual DNA. By using that DNA and Bionatics' simulation technology, you can then reproduce an instance of that tree at any given age and time of the year, including seasonal changes.

Tree modeling is only one of the features found in NatFX. Once the tree is done, the package offers a variety of methods to reduce its triangle counts to real-time requirements. From a full 3D representation of a tree (typically in the tens of thousands of polygons), NatFX will construct a hybrid representation of the same tree, combining geometry and texture maps. The result? The tree shrinks down to the 1,000 or 100-polygon range, while keeping a very similar look to the high-resolution model. Additionally, NatFX can create different levels of detail (LODs) for the same tree. Beginning with the full 3D version for extreme close-ups, you can define up to three LODs of descending triangle counts. By using some clever alpha transitions, you can smoothly move from miles to meters, without perceiving any quality loss.



Aleppo Pine, 80 years old, 92,000 polygons.



Aleppo Pine, 80 years old, 8,500 polygons.

NatFX provides interesting features custom-tailored for game developers: from tuning the number of billboards employed to reusing texture maps to reduce memory footprint, the package really delivers in terms of features. The technology is rock-solid, creating some of the best-looking trees around. They look quite natural, tending away from the classical recursive look found in some packages.

Still, NatFX comes with a few annoying glitches that spoil an otherwise smooth ride. To begin with, the interface is not very intuitive. Ideally, artists would be allowed to work with concepts they are familiar with, such as color, size, and age. Sadly, this is not the case with NatFX. Artists must understand how the program works in order to exploit its full potential. Actions such as setting the desired number of triangles, modifying already existing trees, or changing the color of the leaves would seem logical in a tree-designer mindset. And yet the software fails to present its workflow in a simple, coherent way.

Last, NatFX's advantages come at a price. At a base license of $990 (with 10 plants) and an additional $100 per extra tree, NatFX is a bit on the costly side, especially considering you have no way (other than purchasing trees from Bionatics) to expand your tree collection. There is no editing tool to allow users to create variations or completely new species. So, if you need to create a varied collection of trees, you are certainly going to pay for it.

Still, NatFX is one of the best tree-modeling packages based on features. It promises a lot, and delivers, creating stunning trees that will work well with your 3D engine of choice. It has lots of options and sliders to play with, from LODs to texture reusing and aging, and the geometry reduction method is just fantastic. Still, it's a shame that some interface problems and the pricing scheme prevent it from being the ideal.

★★★½ | NatFX
Bionatics | www.bionatics.com

Dani is the founder and lead programmer of Novarama Studios, an independent developer from Spain. He is also the director of the master's program in Computer Games at UPF (Barcelona, Spain), where he is involved in research projects. He can be reached at dani@novarama.com.

GREAT CIRCLE 6.0.0.9

by francis irving

There's nothing worse than finding you still have a serious crash bug a week before gold master. It happened to me on CREATURES 3, and although it was exhilarating, it was also terrifying that if we didn't fix it we would miss the CD

excellent
very good
average
disappointing
don't bother

factory slot. As serious software developers we need better tools available as standard to end this kind of problem for good. So it was with some interest that I installed Great Circle, Geodesic's testing and diagnostic environment.

At first glance Great Circle appears to be just a memory problem detector. The underlying technology is quite clever; it's really a general-purpose garbage collector for C++ (more about this later); with little performance overhead you can connect it to your game and observe it while it's running.

Great Circle can detect memory bounds overwriting, multiple calls to free on the same piece of memory, and memory leaks. For me the most exciting feature is memory profiling, which would have detected our problem with CREATURES 3.

In contrast to memory leaks, you sometimes get memory used up which is freed when the application shuts down but goes out of control while it is running. A memory profile lets you view allocated memory at any point while the game is running, with a stack trace for each allocation. This information can be used to fix memory drains and reduce the memory footprint.

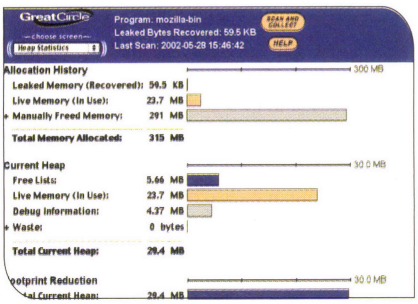
So, how hard is it to use Great Circle? There are two ways of attaching it to your program, either by injection into an existing application, or by recompiling. I had difficulty with both, because when something goes wrong you don't get a decent error message; it just doesn't work. A good knowledge of Visual C++ is essential to help interpret the manual's confusing instructions. There are several tricks you might need to use to get it going. Geodesic's customer support helpline is very good, so when you have trouble they should be able to help you get it on track. Keep in mind that all products of this nature are quite hard to set up.

Once up and running you view information from a web browser. This makes the interface quite familiar and reliable, but also fairly clunky. Errors in the program are quickly highlighted in red and you can click through to a call stack to find the source of the problem.

The quality of information is by and large good, but sometimes there is spurious extra data. For example, memory allocations mysteriously appear with no call stack. It would be useful if Great Circle at least listed the library in which the allocation occurred.

I tried Great Circle out on a game that Creature Labs is about to release which has a memory drain. Unfortunately, the problem turned out to be undetectable by most tools, including Great Circle. The leaking memory was not on the heap but

★★★★★ excellent
★★★★ very good
★★★ average
★★ disappointing
★ don't bother



Great Circle's Heap Statistics function at work.

allocated in a different way by DirectX. Great Circle is very cross platform and has no Windows API– or library-specific features, so it isn't surprising that it didn't catch this. However, there is room in the market for a tool with far more heuristics and which looks at the memory that Windows reports is used by the process, rather than trusting an overloaded malloc function.

Great Circle isn't very well suited to the game industry. It is very cross platform for the business world but is no good for game consoles. It supports injection to help with systems integrators who struggle with legacy applications, when game companies always have the source code. There is, however, a separate companion product using the same technology which may be useful, called Geodesic Runtime Solutions, which is a library that you compile into your released application. If you have missed any memory errors, it compensates for them, both by acting as a C++ garbage collector and by preventing obvious mistakes such as double frees. The garbage collector actually scans memory for pointers which may refer to a block of memory and if it doesn't find any it recycles the memory. This could add an extra layer of stability to your game client or server. It's also the only solution if the memory leak is in a third-party library for which you do not have the source code.

Overall, Great Circle isn't particularly cheap at $1,495. Check out the competition before buying. Having said that, keep the Runtime Solutions in mind; it may just give your next game server that extra fraction of a percent at uptime.

Note: Great Circle will only run under Windows NT, 2000, or XP operating systems. This product is not available for Windows 98.

★★ | Great Circle 6.0.0.9
Geodesic | www.geodesic.com

Francis Irving is a senior programmer making virtual organisms at Creature Labs (www.creaturelabs.com). Contact him at francis.irving@creaturelabs.com, or visit his personal web page at www.flourish.org.

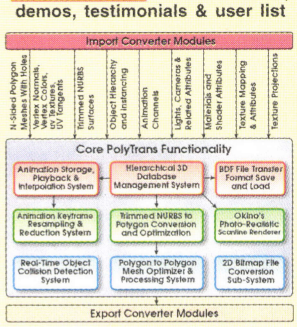
PolyTrans and NuGraf
Convert, View, Render 3D Models & Scenes

Used worldwide by thousands of 3D-industry professionals, Okino's PolyTrans® and NuGraf® are indispensable production pipeline tools that perform clean, robust and reliable file translations (scene, NURBS, CAD and animation data), rendering, material editing, optimization and viewing of the most popular 3D file formats. Download a demo today!

Now with 3ds max™, Maya®, XSI™ & DirectX® Skinning Support!
Email to support@okino.com for full information.

3D Studio® r4
3ds max™ §
Adobe® Illustrator®
Apple® 3DMF®
BioVision® BVH™
Detailer®
DirectX®
DXF®
Form/Z® FACT®
GE2 (Mirai®)
HOOPS® HSF
IGES 5.3‡
Inventor® 2 (SGI)
Inventor® 5 (Autodesk®)
Lightscape®
Lightwave®
Maya®§‡
OpenDWG™
OpenGL® C Code
OpenFlight® (FLT)
OpenNURBS®
Pro/Engineer® .slp
Protein Database®
Renderman® RIB ‡
Renderware® RWX
SAT 8+ (ACIS®)
Shockwave-3D®
Softimage®-3D ‡
Softimage®-XSI™
Solid Edge®
Solid Works®
STL
trueSpace®
USGS DEM
Viewpoint® VET®
VRML™ 1 & 2
Wavefront® OBJ ‡
XGL®(WEB)

www.okino.com for file formats, demos, testimonials & user list



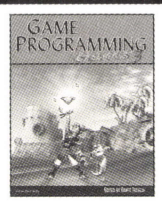
With Maya®, 3ds max® & XSI™ Plugin Support!

Some formats only import or export, or available on Intel only. ‡ = NURBS. § = Requires resident copy of this application.

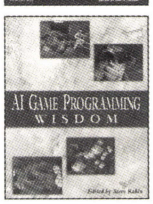
New: BVH™, HSF, Inventor® 5, OpenDWG™, XSI™

Toll Free: 888-3D-OKINO, WEB: http://www.okino.com
T: (1-905) 672-9328, F: 672-2706, Email: sales@okino.com
Okino Computer Graphics
All products mentioned are registered trade names of their respective holders. "apeBot" scene copyright © 2002 by Newtek, Inc. and Matt McDonald of Vision Scape Imaging, Inc.

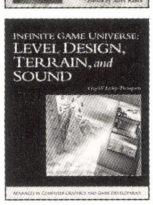
ENHANCE YOUR GAMEPLAY



GAME PROGRAMMING GEMS 3
Contains almost 70 new algorithms from the experts with practical ready-to-use techniques.
Dante Treglia
1-58450-233-9 $69.95 ~610pp



AI GAME PROGRAMMING WISDOM
Provides a comprehensive collection of expert AI algorithms and insights for adding reality to your gameplay.
Steve Rabin
1-58450-077-8 $69.95 672pp



INFINITE GAME UNIVERSE: LEVEL DESIGN, TERRAIN, AND SOUND
Provides a comprehensive guide to level, terrain, sound, and music creation, with ready-to-use source code.
Guy W. Lecky-Thompson
1-58450-213-4 $49.95 ~400pp

GAME DESIGN PERSPECTIVES
Provides a variety of different perspectives from other game designers to use in your designs.
Francois Dominic Laramee
1-58450-090-5 $39.95 410pp

Available at Bookstores Worldwide

www.charlesriver.com
800.382.8505
CHARLES RIVER MEDIA

A New Design Freedom
Coming This Summer
The World's First Completely Programmable Visual Processing Architecture
3Dlabs
Graphics Evolved
A CREATIVE Company
www.3Dlabs.com
3Dlabs is a registered trademark of 3Dlabs Inc. in the United States and other countries.

What a Riot!
A conversation with Chris van der kuyl

Chris van der kuyl hardly fits your brooding-on-the-moors vision of a Scotsman, but he has been working tirelessly to advance the Scottish game development industry, which sports the two crown jewels of Rockstar North (formerly DMA Design) of GRAND THEFT AUTO and LEMMINGS fame, and Chris's own VIS Entertainment, whose recent output includes the cheeky riot action game STATE OF EMERGENCY. Chris founded VIS in 1996 and has nurtured it from a handful of people to the present staff of 120 in three different offices around the U.K. We caught up with Chris in the comfort of the big blue Scottish Games Alliance booth (he's the founder and past chair of the association) at E3 in May.

VIS Entertainment CEO Chris van der kuyl is helping Scotland stand out on the game development map.

Game Developer. I loved STATE OF EMERGENCY at first sight at E3 2001. How did you chart a course to do a good riot game?

Chris van der kuyl. We looked at various ideas of what we could do as a concept. It was at a time before the Seattle stuff — we always get lambasted for going on the back of the WTO riots in Seattle — but we were actually working on it before it all happened. So we thought, O.K., this whole idea of civil unrest is quite cool. Imagine if you put yourself in the center of a riot, how good would that be on a next-generation console if you can literally see hundreds of people screaming and they're all going off doing their own thing?

GD. What did you identify as the central fun factor of the game?

CVDK. The sheer volume of people around. It's the sheer mayhem you can cause with an Uzi. We really purposely went to a kind of comedy, cartoon look.

GD. You don't necessarily want to be too realistic with that kind of situation I guess.

CVDK. Right, because that wasn't the point we were trying to make. It's all tongue-in-cheek. And undoubtedly there's going to be sequels. One of the things we did for the first one is from the minute you start playing the game it is all played at 100 percent. But people said, we'd like to have more depth, we'd like to see a bit of story where you slowly kick off a bit of a disturbance here and see what happens.

GD. Make causing trouble a little more strategic?

CVDK. Absolutely. The AI from that perspective will get a lot more sophisticated. The model is pretty good, we maxed out at something like 400 autonomous characters simultaneously. But we found out that was too many. You couldn't cope with that many people running around. So 100 people is as much as you want to see, because you just couldn't physically cope with such a huge mass coming at you. You can't watch it.

GD. How do you keep growing teams focused on the successful execution of a certain vision for a game?

CVDK. Our teams are running 40 to 45 people, and that's a lot of people to be focused on one game and one idea. If the vision for that game is not clear, then they're going to tamper with it in their own way. That is why the great talent is so good, because they can add things that you never thought about but always wanted. I'm really keen on the idea of having on one sheet of paper not some silly corporate mission statement but how to play the game. If anybody is working on the game, they can always pick this piece of paper up and make a decision themselves.

GD. So you're part of the current anti-documentation backlash?

CVDK. The idea that you can build 500-page document with every piece of the game and expect 45 people on your team to follow it as if it was an instruction manual is wrong. If it truly was an instruction manual for the game, it would take you the same amount of time to write the game as to write the document, so you're talking about a three-year document. If you're going to define every single element so that basically somebody who was untalented could pick it up and implement it, that's like painting by numbers.

GD. How easy or hard is hiring development talent in Scotland?

CVDK. We have people looking at these issues constantly. We do ad campaigns in the U.K. and European trade press. We spend a massive amount of time on graduate recruitment, so we work with the universities and help them develop courses that suit us and suit them, because we know they are places we can go to. We try to get a balance, because you want a good number of young guys coming through but you also want some experience coming in.

Meltdown UK

Microsoft Developer Day

Microsoft®

At the Game Developers Conference Europe 2002
Tuesday, August 27, 2002, Earl's Court
Register now at **http://www.gdc-europe.com**

Plug into the Power of X!

Learn about cutting-edge technologies from the Microsoft DirectX® eXperts. Topics include:

- Surface Material Physics for Shaders
- Shader Optimizations for DirectX
- Power Debugging with Visual C++ 7.0 including the DirectX Programmable Vertex and Pixel Shader Debugger
- Art Pipeline Support and the DirectX 9 High-Level Shading Language

Get the latest on DirectX at
http://msdn.microsoft.com/directx

Microsoft® DIRECTX®

© 2002 Microsoft Corporation. All rights reserved.
Microsoft® DirectX® and Visual C++ are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Rendering Level-of-Detail Forecast

This month I'm going to talk about rendering level-of-detail (LOD) algorithms and make some recommendations about how to use them in the game you're currently designing. When I say "rendering LOD," I mean reducing the number of triangles you use to represent a 3D scene visually at run time, in a way that tries to put most of the geometric detail where it will be most effective. In years past, when game programmers said "LOD" we typically did mean "rendering LOD." But these days we often talk about LOD for a number of different systems, such as AI and physics. So don't be lulled into thinking that if you solve the rendering LOD problem, your game is golden; other LOD-style problems may still be lurking.

For rendering LOD, we have a number of types of systems to choose from. We can use a continuous level-of-detail (CLOD) system, progressive meshes, or the old low-tech solution in which we have a number of precomputed levels of detail for each mesh and we just switch meshes based on viewpoint distance criteria ("static mesh substitution"). I'm going to emphasize the point that you ought to be as low-tech as possible and use static mesh substitution. I'll give more detail soon, but first I will justify my assertion by criticizing the other methods of LOD. I'll start with CLOD, since that's what I have worked with the most.

A Basic Property of LOD

To understand that CLOD schemes are flawed, let's take a look at the basic nature of LOD'd geometry. Figure 1 shows some screenshots from a terrain project I worked on a couple of years ago. Figure 1a shows the rendered image, and 1b shows a wireframe representation of the terrain geometry; I have circled groups of triangles in the scene based on their distance from the viewer. The triangles circled in red are very close to the camera; yellow indicates triangles in the middle distance, and green indicates triangles that are very far away.

In this image, roughly half the triangles on the screen are close to the viewpoint. We expect this from LOD'd geometry, assuming that the world is shaped mostly two-dimensionally (like a terrain, or most first-person shooter maps) and that the idealized geometry does not consist of large, flat surfaces. By "idealized geometry," I mean the geometry of your world at the maximum level of detail, which might even contain infinitely fine features (if your surfaces are fractally shaped, for example).

You can do some simple math showing that this behavior is expected. You start by writing down the constraint that your LOD algorithm uses in order to split triangles (usually, it bounds the screen-space-projected length of some world-space distance, like Lindstrom-Koller's delta values or ROAM's wedgies). Assuming that the average value of this world-space distance is proportional to the length of a triangle's edge (meaning that detail does not diminish with scale), you can see that the LOD algorithm is trying to make the triangles projected to the screen be of roughly the same size.

Then, do some trigonometry: Since the map is mostly 2D, idealize it as an infinite plane being projected to the screen. Imagine a frustum with a 90-degree vertical field of view, so that the vector from the eye through the bottom of the screen is pointing at your feet in world space, and the vector through the top of the screen is pointing toward the horizon. Then, if your eye is a height h from the ground, half the triangles on the screen are within distance h of your feet! Now, a 90-degree vertical field of view is pretty excessive, and you don't usually walk around in such a way that you can see your feet, so this result is a little bit high. But if you play around with the result on a piece of paper, you'll see that the numbers don't get all that much better as you tweak the parameters.

The Problem with CLOD Algorithms

Most CLOD algorithms maintain a persistent mesh that they take great pains to update incrementally, under the misconception that these meshes will not change quickly. As the ROAM paper (see For More Information) says: "ROAM

JONATHAN BLOW | *Jonathan is a game technology consultant who can't think of anything to write in this space. Send suggestions to jon@bolt-action.com.*

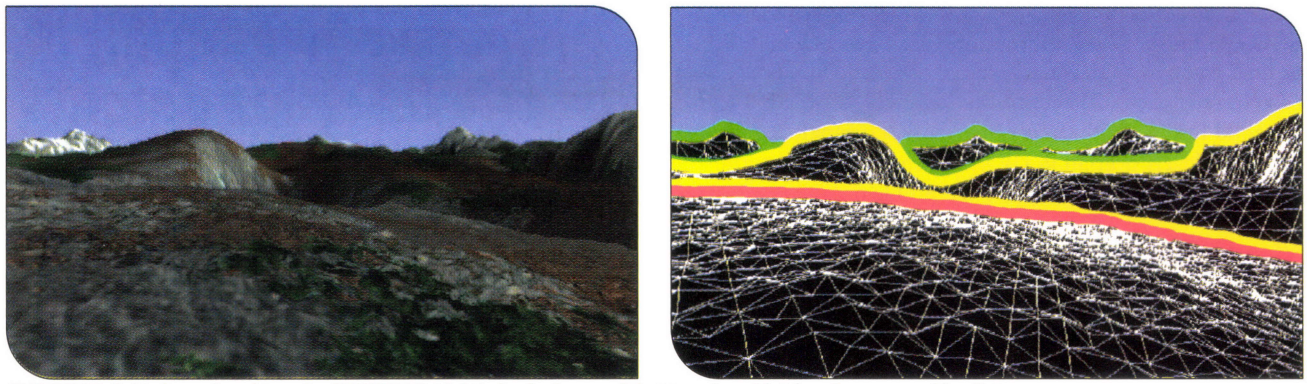


FIGURE 1A (left). A rendered terrain. FIGURE 1B (right). A wireframe of that terrain, with triangles grouped by distance.

execution time is proportionate to the number of triangle changes per frame, which is typically a few percent of the output mesh size, hence ROAM performance is insensitive to the resolution and extent of the input terrain." There's a contradiction inherent in that sentence.

If you, the avatar in a game world, are two meters tall, half your tessellated mesh is within two meters of you. By definition, if you move two meters forward, most of the active tessellation will have to be recomputed. How much time does it take you to walk two meters in real life? How long does it take in a game world, where we often move at unrealistic speeds?

Most CLOD algorithms do a lot of bookkeeping to change the mesh continuously; that bookkeeping is much slower than the clean "discard this, use that" approach of static mesh substitution. When your terrain is detailed enough, and you move quickly enough, these algorithms drown in their own bookkeeping.

These problems don't usually arise in the research papers that present CLOD algorithms, because those papers use relatively low density tessellations, low viewpoint speeds, and viewpoints that are very far from the terrain being viewed (without corresponding enlargement of the viewable terrain distance). But anyone who tries to render a 70,000-triangle CLOD terrain will need to sweat very hard to make the system run acceptably. At the same time, you can render a 500,000-triangle terrain with static mesh substitution, and you don't even need to think much; you need maybe one-fifth of the software engineering effort to get the program working, and afterward, the program is much easier to maintain.

In addition, CLOD systems usually impose topology constraints that we don't want (terrain algorithms require the input data to be a height map, so you can't have overhangs or holes). And I can argue that nobody's even shown that CLOD algorithms provide good tessellation efficiency, which would supposedly have been the whole point of using them. But there's not enough space in this entire magazine for that particular rant.

The only time I can recommend use of a CLOD system is when your data set is constantly changing, in which case you use a nonpersistent form of CLOD, such as Seumas McNally's "split-only ROAM" (see For More Information).

Once upon a time, when I was very enthusiastic about CLOD, I had an e-mail conversation with Charles Bloom and Tom Forsyth. They were proponents of using VIPM, a certain kind of progressive mesh, to represent terrains, but I was convinced that CLOD was better. Well, I've clearly changed my mind about CLOD, but I don't think progressive meshes are very good, either.

The Problem with Progressive Meshes

Like CLOD terrain, progressive meshes have a certain technological wow-factor, because they also provide a continuum of sorts: you can adjust the LOD of a rendered object to a granularity of one triangle. And with certain progressive mesh implementations, like the VIPM implementation shown to me by Tom and Charles, you can render those objects efficiently with modern 3D hardware.

But when rendering an object in a game, such as a character or a vehicle, we just don't need fine-grained control of the object's triangle count. When our total triangle budget is in the millions, small changes in triangle count are completely in the noise. When we're building static levels of detail, the number of triangles we want to discard is usually proportional to the number of triangles in the mesh. If you're building static LOD for a 5,000-triangle object, it makes sense to build a reduced mesh at 2,500 triangles, but building a mesh at 4,900 triangles would be silly.

One might think that the single-vertex-at-a-time path provided by progressive meshes would help with tasks like geomorphing. But it doesn't; when geomorphing, you need to be able to skip across large numbers of collapses, because otherwise you're limiting the speed at which you can adjust the object's detail. This limitation would result in poor system behavior.

So progressive meshes do not provide a concrete benefit over static substitution. But they're more complicated, so they

incur more software-engineering overhead, making your game harder to finish. When approaching problems like building normal maps to augment geometry with static meshes, you can consider each different-resolution mesh in isolation and get good results. With progressive meshes, triangles of different detail levels overlap in UV space, so you have to write more complicated code to solve the problem.

The Plan

So if you're designing an LOD system for your next game, my recommendation is to switch between precomputed static meshes, just like we did in the mid-1990s. Back then, because of the low triangle counts, we would usually have art folks hand-create the low-resolution meshes. These days, there is no good reason to do that. High-quality mesh reduction algorithms, like Garland-Heckbert Error Quadric Simplification, do a good job of generating low-resolution meshes, and your artists' time is valuable.

I've recently been playing around with the normal-map generation technique put to good use in DOOM 3. Using this technique, we can render meshes that appear to have a large amount of geometric detail, but that really consists of a moderate number of triangles. But aside from just looking good, this technique helps make LOD more effective than it has been in the past. LOD is about removing vertices from a mesh, but historically, we have tended to color objects with vertex-based lighting, so the LOD was removing the maxima and minima of our lighting functions. This caused a lot of popping.

With dot3 normal mapping, we can greatly reduce the extent to which the object's appearance depends on individual vertex positions, and thus the popping decreases. This is a case in which static-mesh switching benefits tremendously, yet the more complex CLOD systems find it very difficult to leverage this technique, so they are left further behind.

What about the Game World?

So that's what I recommend for objects, but what about terrains or office complexes?

Casey Muratori and I tend to make the same predictions about future environment LOD. There will be a unification, with the same kind of system used for indoors and outdoors. It'll involve chopping the world up into a bunch of blocks. We then perform a process like mip-mapping, where each of these blocks of geometry is like a texel from a texture map. To get a lower level of detail for the whole world, combine every four neighboring blocks and detail-reduce the resulting mesh (four blocks if the world mostly extends through two dimensions; it should be eight blocks if the world is more volumetric). Keep doing this until you get all the way down to one block representing the entire world.

At run time, you start near the camera, rendering high-resolution blocks; then, as you traverse the scene and go farther from the viewpoint, you switch to lower-resolution blocks. At places where the resolution transitions occur, you will need to stitch together blocks of neighboring detail levels, or else cracks will appear in the scene. This stitching is trivial for the case of a height map; it becomes more difficult with arbitrary topologies crossing the detail boundaries, but the solution is largely a matter of bookkeeping. Since your algorithm performed the detail-reduction steps for the blocks, it can effectively trace backward along that sequence of reductions to see how edges should match up.

Thatcher Ulrich implemented something like this procedure for terrain rendering. Sometime soon, I plan to do it for triangle soups; drop me a line if you beat me to it.

When we're building static levels of detail, the number of triangles we want to discard is usually proportional to the number of triangles in the mesh.

FOR MORE INFORMATION

Duchaineau, Mark, and others. "ROAMing Terrain: Real-time Optimally Adapting Meshes." *Proceedings of IEEE Visualization, 1997.*

McNally, Seumas. "The Tread Marks Engine," part of "Two Advanced Terrain Rendering Systems." *2000 Game Developers Conference Proceedings.*

The Virtual Terrain Project
http://vterrain.org

Ulrich, Thatcher. "Chunking LOD"
http://tulrich.com/geekstuff/chunklod.html

Garland, Michael, and Paul Heckbert. "Surface Simplification Using Quadric Error Metrics." *Siggraph 1997.*
http://graphics.cs.uiuc.edu/~garland/research/quadrics.html

Put Your Game Online
We'll Keep It There

Windows® * PlayStation® 2 * Nintendo GameCube™ * Xbox™ * Macintosh® * Linux®

gamespy.net

We'll take care of everything: technology, servers, support, even bandwidth costs.

Getting rock-solid player matchmaking, chat, and server browsing in your game is only the first step. When you work with GameSpy, you get more than solid, easy-to-implement technology. You also get reliable multiplayer back-end support, 24/7, for the lifespan of your product.

We've got over six years of multiplatform online gaming experience. You make the game, we'll keep it running.

Find out more:
www.gamespy.net
bizdev@gamespy.com
Art Santos - (949) 798-4267

POWERED BY gamespy™

Technology to put your game online.
Service to keep it there.

Online solutions for Windows®, PlayStation® 2, Nintendo GameCube™, Xbox™, Macintosh® and Linux®

- Peer-to-Peer Player Matching
- Server Browsing
- Instant Messaging and Buddy Lists
- Text and Voice Chat
- Online Storage of Game Data
- Competition and High-Score Systems
- Online Anti-Piracy
- Usage Analysis and Reporting
- Cross-Platform Networking
- Auto-Patching
- Online Data Transfer

Copyright © 2002 GameSpy Industries, Inc. GameSpy, Powered By GameSpy and the SpyGuy Logo are trademarks of GameSpy Industries, Inc. , All rights reserved.
All other trademarks and names are properties of their respective owners.

Building the Future Part 2: Vehicles

So, what is the future going to be like? Will we find ourselves booking an operation to implant the latest biomechanical eyes from Nikon, courtesy of a William Gibson future of cybernetics? Will we end up being chased through a decaying hotel by a Scandinavian android in a Philip K. Dick–style dystopia? Or (and this is my personal favorite) will we all be running for our lives from impossibly large insects that have been mutated by mankind's careless use of nuclear energy?

The best thing about the future, from a designer's point of view, is that it could turn out to be pretty much anything. Setting a game in the future takes advantage of this flexibility and gives the artist a bit of space to be inventive. There are, however, some areas that can benefit from an element of realism if readers are to read the game successfully.

Long ago, in "Building the Future, Part 1" (January 2002), we looked at how to envision the future through architecture. This time, we'll look at designing our way around in our futuristic environments, in vehicles.

The best thing about the future, from a designer's point of view, is that it could turn out to be pretty much anything.

Simplification or Complexity?

The history of industrial design tells an interesting story. Looking at the last 60 years, we can follow a path that has firmly linked design to technological advancement. It is interesting to examine the progression that has led from rudimentary concepts built to be functional, through the era of "high-tech" knobs and buttons (1970s and 1980s), into what we now see as desirable: simplicity and attractiveness.

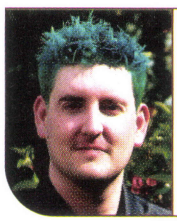
Perceptions of what "high-tech" actually means have changed throughout the decades; this, in part, has influenced designers keen to produce progressively more cutting-edge designs. These perceptions, mirrored in the science-fiction films of each period, give us some insight into what represented futuristic transport in each era. The relatively simplistic forms in *Forbidden Planet* (1956) become the complex industrial shapes of *Alien* (1979), that once again move back toward simplified shapes of modern-day *Star Trek*. The technological aspect is represented as less industrial and more aesthetic.

There are many ways in which the technology of the future may advance: miniaturization, and a more complete understanding of the laws of physics and the development of new materials together with their associated methods of fabrication, are possible future trends. An excellent illustration of the impact of miniaturization (through advancements in electronics) can be found in studying the development of the modern calculator.

While the rules of addition and subtraction have not changed, the advent of the transistor and the subsequent rampage of microelectronics have shrunk the size of calculators significantly as each decade has passed. We are now able to create a complex calculating circuit on the head of a pin, and this projected level of miniaturization allows us to extrapolate into the future and make assumptions about our vehicles and their technology.

In terms of designing vehicles, areas such as aerodynamics and advances in fabrication processes have made forms previously seen as purely theoretical and made them realistic.

Vehicles of the 1920s, such as the Model T Ford, were limited by the materials and construction methods available at the time, as well as levels of understanding



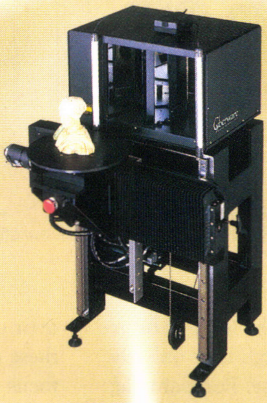
HAYDEN DUVALL | *Hayden started work in 1987, creating airbrushed artwork for the games industry. Over the next eight years, Hayden continued as a freelance artist and lectured in psychology at Perth College in Scotland. Hayden now lives in Bristol, England, with his wife, Leah, and their four children, where he is lead artist at Confounding Factor.*

3D scanning...

...for Complex Objects

DESKTOP 3D SCANNER

High resolution, compact:
- Sculptures
- Models
- Bones



HIREZ MINI MODEL SHOP COLOR 3D SCANNER

High resolution, mid-size models:
- Design
- Prototypes
- Reverse Engineering

MODEL SHOP COLOR 3D SCANNER

Standard resolution, large models:
- Large sculpture
- Maquettes
- Industrial Models

...for Human Subjects

HEAD AND FACE COLOR 3D SCANNER

Standard resolution, composite human face or head:
- Animation
- Engineering
- Design
- Virtual Reality

WHOLE BODY SCANNERS (WB4 & WBX)

Full resolution, whole body scanning:
- Animation
- Fashion
- Ergonomics

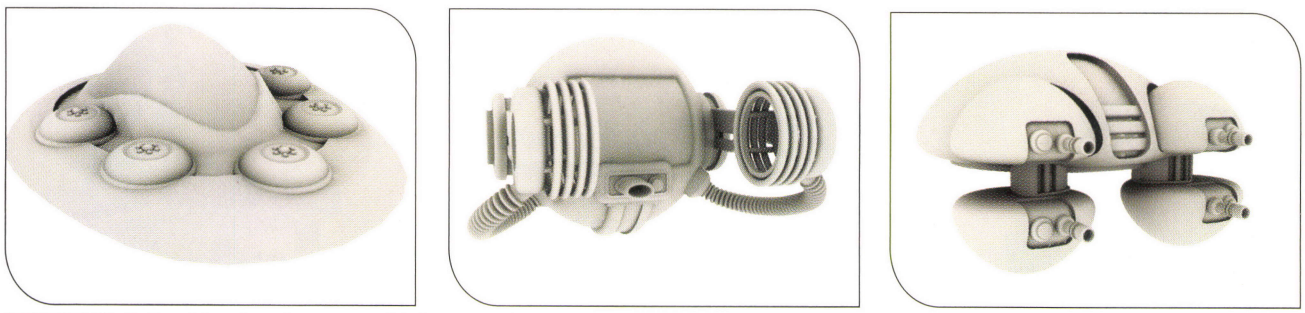
Whatever You're Scanning,
Cyberware provides an entire line of 3D scanners that are fast, accurate, and easy to use.

CELEBRATING 20 YEARS OF 3D SCANNING!

Cyberware®

www.cyberware.com

email: info@cyberware.com



FIGURES 1A–C. Consideration given to specific features of a vehicle help identify its type and function: aerodynamics for racing (a), mechanical devices for maintenance (b), weaponry for combat (c).

in areas such as aerodynamics. By contrast, today's concept cars look ahead to what the future may hold, using our present levels of technology as a springboard.

Assuming that the future will be even more open in terms of vehicle design, which path, as artists, do we choose to follow?

The Mechanics of Motion

As with most design decisions where a future or fantasy world is concerned, you have a choice as an artist: either throw reason out the window and develop a world that is pure fantasy, or attempt to ground your design in something more realistic.

Future vehicles are a pillar of science fiction, and whether they be the oddly asymmetrical *Millennium Falcon* or the long-legged mechanical turtle that was the AT-AT, we can use such vehicles to learn about the world to which we have been transported.

When designing a vehicle, the first (if somewhat obvious) question to ask is, what kind of vehicle is it? Or more specifically, on or through what medium does it travel? When you're considering designing something that travels outside a planet's atmosphere, you obviously have a whole different set of constraints to consider than when you're designing something that travels through water.

TV and films are often guilty of producing ships that have the look of futuristic airplanes even though they only fly over the vacuum of space. For example, the sleek lines of many *Star Trek* ships belie their deep-space designation. The original mining ship from *Alien* (the *Nostromo*), however, neatly illustrated the practical differences between space flight and what one would need to enter an atmosphere. The detachable front module was designed to land on a planet's surface, leaving the unwieldy cargo section in orbit. For the purpose of this article, though, we will leave the vagaries of space travel behind and concentrate on something that's more down to earth: land transportation.

How Much Detail?

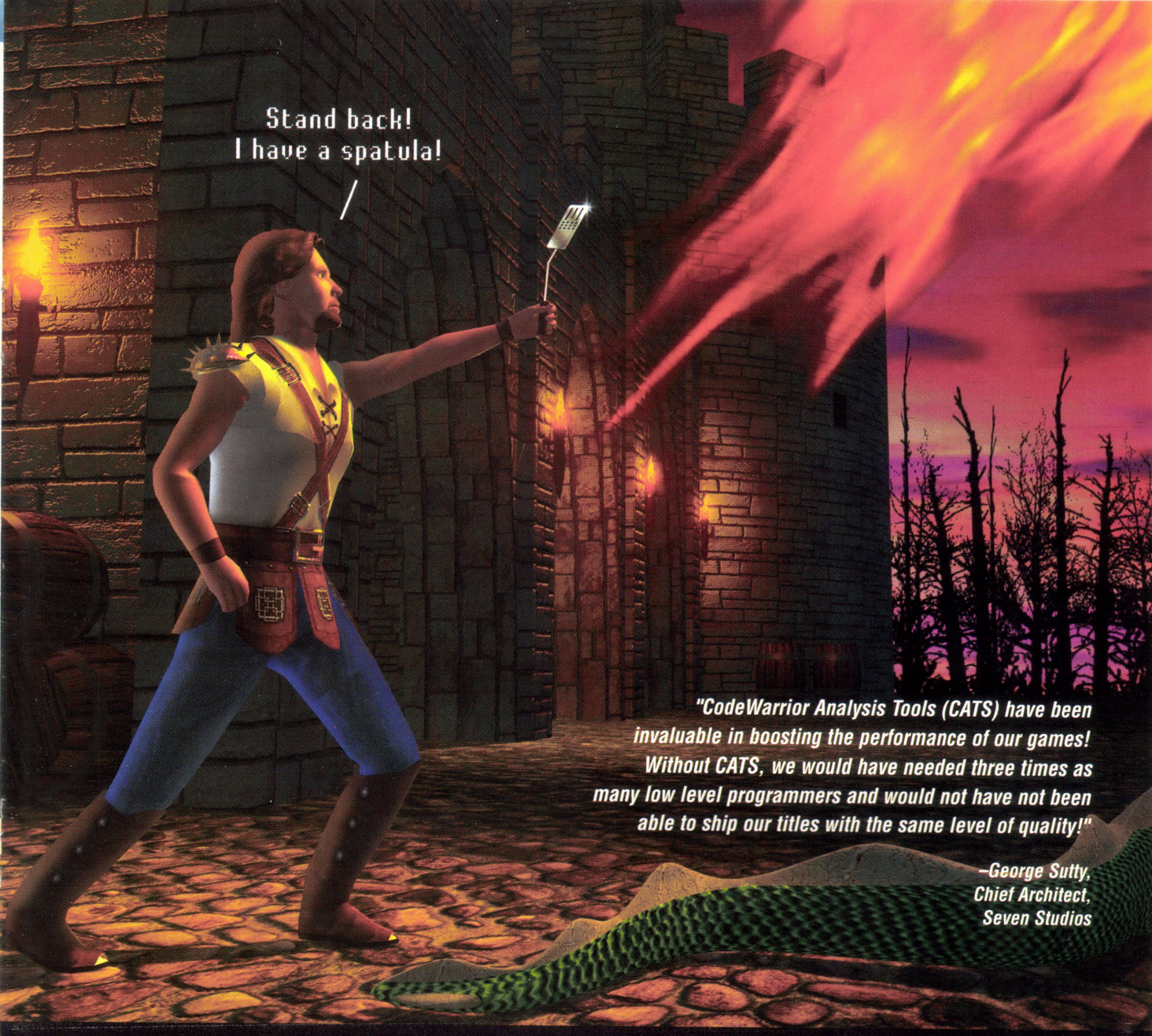
Before laying down a single polygon, it is important to look at your game and establish the level of detail that is appropriate for the vehicles. As always, the platform on which you are developing, as well as your engine's limitations, have a part to play. But possibly the most important element to consider is to what extent the vehicles that you will be making feature in your game.

The most recent incarnations of our favorite racers boast cars that are immaculately modeled and textured replicas of the real-world cars they represent, right down to the smallest detail. Look at the wheel arch of any of these cars and you'll be hard-pressed to spot the angularity of their constituent triangles. Obviously, as these models are pretty much the entire focus of such games, their developers can safely invest resources on realism. Also, a racing game places much less emphasis on its environment — not that the scenery in a racing game isn't detailed, but the limited area of play means you don't need to create a detailed world, ready to be explored.

In such games as HALO or GRAND THEFT AUTO 3, vehicles still feature prominently in the game's structure, but unlike pure racers they are one element in a much more complex world. Games of this nature need to expend significant energy on character animation and thus aren't able to be as detailed with their models. Fortunately, with games of this kind, a player's attention is going to wander from the vehicle toward the marauding aliens or the machine-gun-toting thugs, and so absolute accuracy of detail becomes less important. Tailoring the level of detail of your vehicles to make sensible use of resources in the context of your game design is always a worthwhile exercise.

Fight or Flight?

We also need to consider the vehicle's designation. Will it be used for combat, racing, civilian transport, or turnip harvesting? A vehicle's shape is largely defined by its functionality. But while it is certainly true that basic vehicle shape is normally linked to function,



Without the right tools, You're Screwed.

Does your game have you in a world of pain? Turn to **CodeWarrior Analysis Tools** from Metrowerks–the leader in game development software. Our performance-enhancing tools give you a secret weapon for the entire game development process. So that your games aren't just faster, they're also faster to market.

Plus you can rely on Metrowerks to be there for one-on-one support. To get the tools you need and find out about our latest promotion, call **1.800.377.5416**, or visit **www.metrowerks.com/games.** And let the games begin.

CodeWarrior™
Game Development Tools

NINTENDO GAMECUBE®
tools and middleware vendor

Tools & Middleware PlayStation®2

© Copyright. 2002. Metrowerks Corp. All rights reserved. Metrowerks, the Metrowerks logo, and CodeWarrior are registered trademarks of Metrowerks Corp. in the US and/or other countries. NINTENDO GAMECUBE is a trademark of Nintendo Co., Ltd. PlayStation is a registered trademark of Sony. Inc. All other tradenames and trademarks are the property of their respective owners. All rights reserved.

metrowerks
Software Starts Here

In terms of designing vehicles, areas such as aerodynamics and advances in fabrication processes have made forms previously seen as purely theoretical and made them realistic.

often additional features help define a vehicle's type.

Figure 1 shows how a combination of overall styling, together with the addition of aerodynamics, mechanical devices, or weaponry can set a vehicle up as being for racing, maintenance, or combat.

Propulsion

When considering what it is that makes an individual vehicle futuristic, a good place to start is propulsion, or how a vehicle is powered. One area available for an artist to convey advances in design is in imagining how a method of transport may be powered in the future. The legendary concept artist Syd Mead (*Tron*, *Blade Runner*) has said that part of his design process is assuming that some leap in technology has occurred between now and the future point for which he is designing, and one of the most obvious ways in which this assumption can manifest itself is propulsion.

Most ground-based vehicles use wheels, and while it's reasonable to suggest that there's nothing especially futuristic about wheels, you can use wheel design, sizing, and number of wheels to build the impression of a futuristic vehicle.

Also, tracks are a reasonably recent addition to vehicle design and generally represent a vehicle built for difficult terrain, most often military. However, the juxtaposition of tracks and traditionally wheel-driven vehicle types can produce interesting results if this fits in with your overall design.

As we get more futuristic, we inevitably see vehicles that are powered by a form of jet engine or have some kind of antigravity hovering capabilities. The hover car has seen many incarnations, but the central idea of some form of antigravity device is appealing, as it requires no obvious mechanism, as jet-powered vehicles do. However, even when we eliminate friction through hovering, we need to take care of forward propulsion, and here again, some form of jet engine is important.

The Need for Speed

Speed is often perceived as one of the main advances we can expect from travel in the future. This prediction is not without foundation, as even though speed limits on our roads have more or less remained the same for the past several decades, top speeds of vehicles produced have gone through the roof, with most regular production cars reaching 120 mph and beyond.

Several visual features can have an impact on conveying the idea of potential speed (besides painting flames along the sides). They are somewhat in opposition, but depending on the style constraints of your game, one or both methods may be useful.

The raw power needed to push a vehicle to extreme speeds is often conveyed successfully by oversized engine parts or excessive means of propulsion that seem insane in comparison to the vehicle they are powering. This approach favors the cyber-punk design ethic, championed by the *Mad Max* school of automotive design and exemplified in the insanity of *The Phantom Menace*'s pod racers. If we are going to travel fast in the future (and most of us would be disappointed if this weren't the case) we will need some serious machinery to propel us. Building this kind of model for a game can require more geometry, with tubes and pipes, air vents and exhaust ducts. However, it also allows the artist plenty of room for invention, and much of the finer detail can be put in texture.

The alternative to this approach is to take the more elegant, sophisticated route. Real speed, the kind that accelerates you so fast that it can peel the enamel clean off your teeth, requires more than just welding the mother of all engines to four wheels and a gear stick. If we consider those rocket-shaped cars used to test land-speed records as an example, faster travel also creates the need for better lines, maximized airflow, and the reduction of drag. Extreme speed demands very smooth lines, and this is another possibility to explore for future vehicles.

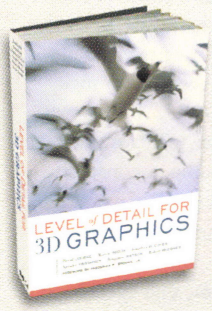
Simpler shapes with less additional geometry to clutter their surface make smaller demands on resources; on the other hand, a mesh that attempts to simulate smooth curves is going to need to be highly detailed to avoid angular edges, so the real savings may be deceptive.

Whichever approach you favor (or even a combination of the two), vehicle design can be one of the most enjoyable parts of an artist's job. As long as you take the time to consider context, designation, and overall game styling, the results can be as spectacular as they are varied.

MK®
MORGAN KAUFMANN PUBLISHERS
An Imprint of Elsevier Science

Visit us on the web at www.mkp.com

PORTABLE TECHNOLOGY



LEVEL OF DETAIL FOR 3D GRAPHICS

Application and Theory

David Luebke | Martin Reddy
Jonathan D. Cohen | Amitabh Varshney
Benjamin Watson | Robert Huebner

"This fine book...provides both a treatment of the underlying theory and a valuable practical reference for the graphics practitioner."

— **From the forward by Dr. Fredrick P. Brooks, Jr.**
UNC Chapel Hill

"....it's exactly what I would want to find on the shelf if I needed to implement some LOD techniques in a game."

— **Mark DeLoura,** *Manager of Developer Relations*
Sony Computer Entertainment America

"This book will find a choice spot on my bookshelf."

— **Will Schroeder,** *Co-founder, Kitware, Inc.*

"The book does a great job of piecing together years of work and presenting it in an organized fashion.

— **Daniel Aliaga,** *Computer Graphics Researcher*
Bell Laboratories

"A fantastic and well-written book, filled with a lot of very useful information. There is no other like it on the market."

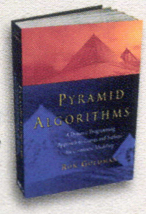
"This is the first and only book that provides a comprehensive coverage about level of detail.

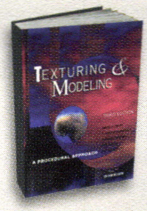
"This book is packed with information. When you are finished reading it, all you will be able to say is Wow!"

— **Dave Eberly,** *President, Magic Software, Inc.*

Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation
Thomas Strothotte and Stefan Schlechtweg

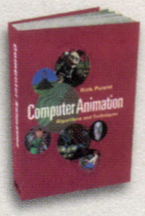
Pyramid Algorithms: A Dynamic Programming Aproach to Curves and Surfaces for Geometric Modeling
Ron Goldman

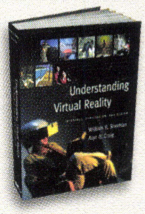




Texturing & Modeling: A Procedural Approach, Third Edition
David S. Ebert, Kenton F. Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley

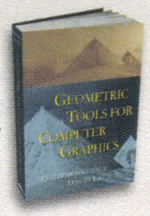
Computer Animation: Algorithms and Techniques
Richard Parent





Understanding Virtual Reality: Interface, Application, and Design
William Sherman and Alain Craig

Geometric Tools for Computer Graphics
Philip Schneider and David Eberly



Game Physics
Dave Eberly

ORDER INFORMATION

Order Fulfillment Department, Elsevier Science, 11830 Westline Drive, St. Louis, MO 63146
Phone: (800) 545-2522 • Fax: (800) 535-9935 • Email: orders@mkp.com

Blending the Total Soundscape

Imagine this: The phone is ringing, a television is blaring in the next room, water is running into a growling garbage disposal, dogs are barking, a neighbor's stereo is thumping, another neighbor has his leaf blower set on gale-force, kids are screaming, skateboards are grinding on the curb, a motorcycle is revving on the street, all while you are trying to enjoy a quiet Saturday morning of leisurely cartoon watching. This is an example of what a bad game can sound like, the kind of horrendous audio mess where so much is happening that you end up missing out on what is ultimately important.

A saturated soundscape means degraded effectiveness. The only options available are to turn it down or turn it off — we can prevent either from ever happening. Music, sound effects, and narratives can coexist within the soundscape peacefully, but a relative amount of care should be taken to ensure they do.

Musical elements. A composer's instincts dictate that each piece of music should be able to stand on its own. In order to make the music interesting, composers add several layers of musical ingredients: a melody here, a countermelody there, percussions, fills, and on and on. If it doesn't sound great on its own, it won't sound good in the game, right? Well, there are appropriate times where this type of music may work: intro sequences, cinematics, menu screens, and other stand-alone cues. However, music intended as background music, has to be just that — in the background. Busy, complex scores do nothing for the overall soundscape when all those musical layers start to interfere with whatever else may be going on.

To compose music that is true to the game is a unique challenge. If music is the only audio playing at a particular time, you are home free. But if the music will be sharing real estate with other sounds, lay back a little and make some room for the rest of the audio to be heard. Save the elaborate performances for a soundtrack release and let your professionalism shine through for the sake of the product.

Sound effects. Compelling sound effects sometimes require a bit of ingenuity. Because the player needs the feedback these audio cues provide, they need to be heard. For something as basic as a menu screen, for example, a jamming dance cue could be playing, psyching up the player for the upcoming experience. The player clicks a selection, and except for the visual cue of seeing the button depress and the screen changing, the player receives no aural response. The sound effect associated with the button click was either too low in volume or lost in the music. Some choices need to be made.

Do you turn down the volume of the music? Do you increase the volume of the sound effect? Do you dump the sound effect? A better solution is to create a sound that actually stands out from the music.

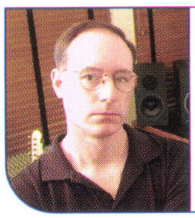
A bass-heavy music track won't leave room for any other low-frequency activity in the soundscape. Hits or explosions may barely be heard because they compete directly with the bass components of the music. To remedy this, consider keeping the sound effects in the mid- to upper-frequency ranges or remix the music with less bass. A "thump" may not be heard among music with this type of low-frequency congestion — try a "click" instead. Conversely, that "thump" will work well with music with a lot of high-frequency, percussive activity, where the "click" might get lost.

Another trick is to build room in the frequency spectrum of the music to let the sound be heard. Set the musical bass elements to reside around the 50Hz range, and explosions around 60Hz. By separating their predominant frequencies slightly and manipulating the audio accordingly, the game can be experienced as intended. Notice how the vocals in a song stand out? Same theory.

Voice-overs, narratives, and speech. Vocal ingredients require the same consideration as other audio. Most game creators have a tendency to want their vocalizations full-bodied, like those attention-demanding FM radio jocks. While these narratives may sound great on their own, when added to the mix of music and sound effects, such frequency-saturated voices become mud. Instead of simply increasing the volume of the speech, narrow the spectrum the voice uses and develop a good mix with equalization. The words will be easily heard and understood, and the rest of the audio will be clear.

Put it all together. A common mistake inexperienced music producers make when doing final music mixes is to solo each instrument and tweak them individually to perfection. When all of the instruments are brought together, however, the resulting competition for space creates a mix that is cloudy.

What ultimately matters, for the good of the overall production, is that everything sounds good together. Applying a little attention to detail allows each audio ingredient to work together, hopefully to be heard clearly and as intended.



AARON MARKS | *Aaron (aaron@onyourmarkmusic.com) is a composer, sound designer, author of* The Complete Guide to Game Audio *(CMP Books), and the humble proprietor of On Your Mark Music Productions (www.onyourmarkmusic.com). He is currently hard at work on game projects for Vivendi/Universal, Flipside.com, and Enemy Technology.*

GameDevelopers Conference Europe

27-29 August, 2002
Earls Court Conference Centre • London, UK
www.gdc-europe.com

Make better games

The Game Developers Conference Europe 2002 focuses on real world problems and solutions. Sessions allow the exchange of hard earned knowledge and practices between the game industry's most successful developers.

Topics include

- Build or Buy: The Muddle of Middleware
- Where is Technology Going? A Panel Discussion
- Critical Analysis of Games: Panel & Debate
- Ratings and Censorship: Making Sense of Videogame Ratings
- Developing in Developing Countries
- Publisher Developer Relations: Bridging the Gap
- Postmortems on:
  - Scripting in BLACK & WHITE
  - AI in STATE OF EMERGENCY
  - Production of PROJECT GOTHAM
  - Audio in THE GETAWAY

Keynote

Preproduction: Myth vs Method
Mark Cerny Cerny Games

Speakers include

Simon & Dene Carter Big Blue Box
Sarah Chudley Bizarre Creations
Charles Cecil Revolution Studios
David Braben Frontier Developments
Francois Masciopinto Infogrames
Glenn Corpes & Jeremy Longley Lost Toys
Jonny Dobson VIS Entertainment
Jonty Barnes & Richard Evans Lionhead Studios
Marek Spanel Bohemia Intertactive
Phil Harrison Sony Computer Entertainment Europe

Sponsored tutorials include

- ATI Mojo Day
- IGDA Academic Day
- Meltdown UK: Microsoft DirectX Developer Day
- Xbox Unplugged

For a complete listing of topics and speakers, visit www.gdc-europe.com

in partnership with ects 2002 29-31 august earls court london

Asymmetrical Distribution

This month's game design rule was inspired in part by a discussion with Teut Weidemann, head of Wings Simulations (developers of PANZER ELITE) in Germany.

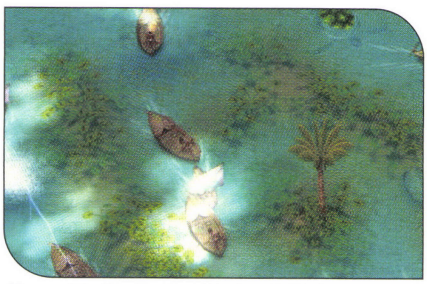
The Rule: Distribute game assets asymmetrically.

When there are objects or experiences the player can encounter in a game, place them asymmetrically, both spatially in the sense of clumping some together and spreading others thinly, and temporally in the sense of having some be common, some uncommon, and some rare over time. Of course, particularly useful or powerful items are good candidates to be the rarest.

The Rule's domain. The rule applies to all games, and in fact, applies to most things the player can encounter in those games.

Rules that it trumps. Asymmetrical distribution trumps rules that emphasize consistency and uniformity in games. It's important, for example, to provide consistent stimulation and an unending flow of interesting decisions for the player to make, but that doesn't mean a predictable and unvarying distribution of the same type of decisions. You can always provide a consistent flow of different kinds of decisions. Since another, closely related rule is "Avoid player fatigue," this asymmetry rule is a good one to follow to keep the distribution of power-ups or other assets uneven and surprising and to keep the player interested and alert.

Rules that it is trumped by. This rule is trumped by the need to provide a smooth learning curve for the game on a larger scale. New game interface concepts, characters, units, and player abilities should be presented smoothly throughout the game so players can master one before being confronted with the next. It would not do to asymmetrically clump the introduction of four new military units in one part of one level of an RTS and not introduce any new ones for the next three levels. But within a level it's a good thing to distribute the resources in clumps, like a forest of trees instead of individual trees spread throughout the landscape, or a gold mine instead of small packets of gold evenly placed in the world.



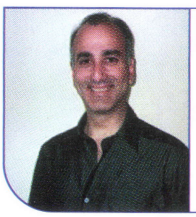
Ensemble Studios' upcoming RTS, AGE OF MYTHOLOGY, shows how resources can be distributed asymmetrically to provide interest.

Examples and counterexamples. This is a very universal rule, understood by designers everywhere. In every good game you will find examples, like areas where there are few enemies followed by an ambush where many descend on you at once, or a few paltry gold pieces followed by a treasure chest full of riches. Richard Garfield turned the use of common, uncommon, and rare items into an entire mini-industry with MAGIC: THE GATHERING. Think how much less compelling the metagame of collection would be if all the cards in that game were equally distributed.

Less often is this principle employed for intangible parts of games. Consider the vocal acknowledgement that an infantryman in an RTS game gives when you click on him. If he says "Yes, sir?" every time, it not only becomes annoying quickly, but also breaks the "Maintain suspension of disbelief" rule (May 2002). If he says "Yes, sir?" or "At your command" or "My orders?" or "You rang?" each 25 percent of the time, it's somewhat of an improvement, a bit more realistic and less monotonous. But if he says the first two things 40 percent of the time each, the third one 15 percent, and the last one 5 percent, a player is likely to be surprised when he or she hears "You rang?" for the first time 10 or 20 minutes into the game.

Adding one more unexpected response such as, "Yeah, whaddaya want? Oh, it's you, sir!" that occurs only 1 percent of the time not only provides a fun surprise for players, but also leaves them expecting to find more unusual utterances and will keep them intrigued long into the game. One old Infocom game, SPELLBREAKER, used a similar principle to hide the developer's favored name for the game that had been vetoed by the publisher. Once in every 100 or so games it displayed the old name of the game, MAGE, surprising quite a few players.

Update to "Name That Trump!" challenge. Last month I set forth a challenge to solicit rules that trumped the rule of allowing the player to save games anywhere. I've received some mail strongly in favor of more games following this rule, but only one trump so far. I'll suggest some answers next month.



NOAH FALSTEIN | *Noah is a 22-year veteran of the game industry. You can find a list of his credits and other information at www.theinspiracy.com. If you're an experienced game designer interested in contributing to The 400 Project, please e-mail Noah at noah@theinspiracy.com (include your game design background) for more information about how to submit rules.*

NEW RESEARCH FROM THE GAMA NETWORK

Getting in the Wireless Game

Opportunities and Entry Points for Game Developers

With projections of mobile gaming reaching global revenues of $5 billion by 2005, you can't afford to ignore this emerging market.

Get accurate, in-depth analysis of this market's technologies and momentum in this comprehensive research report:

- ✔ Identifies key players in the mobile market
- ✔ Explains the varied paths of entry for game developers
- ✔ Includes an overview of wireless platforms and compares capabilities
- ✔ Helps PC, console and web-based developers find the best way to take content mobile
- ✔ Explains the business and partnership opportunities between content creators and media companies
- ✔ Examines emerging opportunities for game developers

Purchase this report at
www.gamasutra.com/store
using code GDMAGWIRELESS02 to

SAVE $400

Regular price: $795 ***Your price: $395***

mbusiness RESEARCH
INSIGHT ACTION ADVANTAGE

GamaNetwork

naty hoffman & arcot j. preetham

Photorealistic Real-Time Outdoor Light Scattering

Some of the most striking aspects of outdoor scenes are the result of light interacting with the atmosphere: shades of blue in a clear noon sky; the red and gold colors of sunset; the purple tint of distant hills; the gray, washed-out look of a foggy day.

In this article, we will explain the basic principles of scattering physics and use them to derive a scattering model. We will then show how to implement this model with a vertex shader, so that these effects can be generated and changed in real time.

Scattering Fundamentals

We will start with some fundamental concepts as a backgrounder:

Radiant flux (Φ) measures a quantity of light through a surface (through all points and in all directions). Radiant flux is power, measured in watts.

Solid angle (ω) measures a sheaf of directions in 3D, like an angle measures a sheaf of directions in 2D. While angles are defined as arcs on a circle and measured in radians (2π in a complete circle), solid angles are defined as patches on a sphere and measured in steradians (4π in a complete sphere).

Radiant intensity (I) measures a quantity of light through all points in a surface going in a single direction. Radiant intensity is power over solid angle, measured in watts per steradian.

Radiance (L) measures a quantity of light in a single ray (through a single point in a single direction). Radiance is power over (area times solid angle), and is measured in watts per steradian per meter squared. The pixel values of the final rendered image are derived from the radiance values for rays going through each pixel into the camera.

Pixel values are RGB triples. However, radiance is distributed along a continuous range of frequencies. There are two ways to derive RGB values from a set of wavelength-dependent equations. The fast way (commonly used for real-time graphics) is to plug three sample frequencies into the equations, resulting in an RGB triple. The more precise way (often used for offline rendering) is to use several dozen samples evenly distributed throughout the visible spectrum. The resulting series of numbers is converted to an RGB triple via perceptual weighting and integration.

Image by Solomon Srinivasan

NATY HOFFMAN | *Naty has been leading the development of the EARTH & BEYOND graphics engine at Westwood Studios since 1997. Previously he worked at Intel as the lead microprocessor architect for the Pentium with MMX chip and contributed to the MMX, SSE, and SSE 2 instruction sets. Contact him at naty@westwood.com.*

ARCOT J. PREETHAM | *Preetham (preetham@ati.com) is a software engineer working on various rendering techniques for next-generation graphics hardware at ATI Research. Prior to this, he developed 3D modeling and reverse-engineering software at Paraform, and worked on rendering atmospheric effects for flight simulators at Evans & Sutherland.*

We will use the fast method, but at a cost. Figure 1 shows the spectral sensitivity for the three kinds of cones in the human retina. We can see that no matter which three sampling frequencies we pick, we will lose information on the spectral structure between them, which introduces inaccuracies.

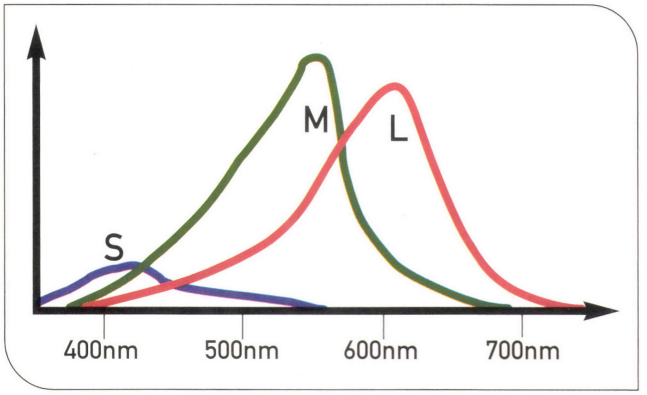


FIGURE 1. Spectral sensitivity curves for cones in the retina.

Atmospheric Light Scattering

There are three types of interactions that can occur between a photon and a particle (an atom, molecule, dust speck, water droplet, and so on). The particle may scatter the photon into the line of sight (in-scattering), it may scatter it out of the line of sight (out-scattering), or it may absorb the photon altogether (absorption).

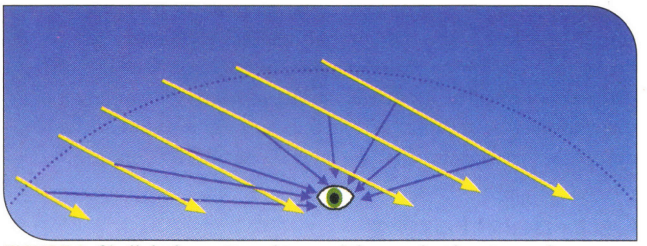


FIGURE 2. Skylight is scattered toward the eye by the atmosphere. Because blue light scatters more than red light, the sky usually appears blue.

Atmospheric light scattering (we include both scattering and absorption under this term) is responsible for many varied visual effects in outdoor scenes, but for the purposes of this article we will concentrate on three: the sky, sunlight, and aerial perspective. First we will discuss the sky. When looking at a clear sky, we would see nothing but black if atmospheric light scattering were not present. In Figure 2 we can see how the atmosphere scatters sunlight toward the eye. Since blue light tends to scatter more than red light (more on this later), the sky usually appears blue.

In Figure 3 we can see how the atmosphere, via out-scattering and absorption, removes part of the sunlight before it reaches the eye. Again, mostly blue light is affected, which causes the

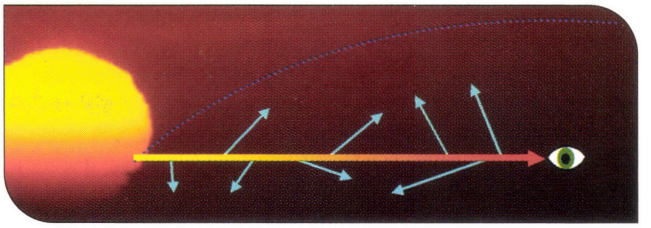


FIGURE 3. The atmosphere removes some sunlight before it reaches the eye by out-scattering and absorption. The scattering of blue light makes the sun appear red when it's near the horizon because its light must travel farther through the atmosphere to reach the eye.

color of the remaining sunlight to shift toward yellow and red. When the sun is near the horizon, sunlight travels a much larger distance through air than when it is at the zenith. This explains why this effect is strongest at sunrise and sunset.

Aerial perspective causes distant objects to shift in color. In Figure 4 we can see how the atmosphere attenuates the light from distant objects via out-scattering and absorption, and adds new light via in-scattering. Since mostly blue light is involved, this causes distant dark objects to appear blue and distant bright objects to appear reddish.

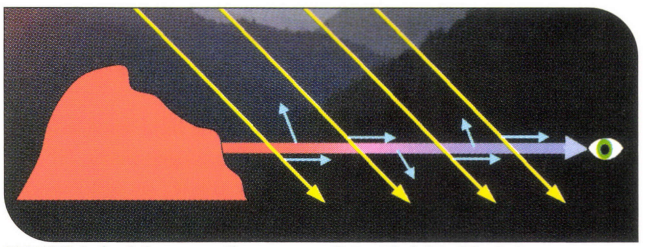


FIGURE 4. Aerial perspective causes distant bright objects to appear reddish and distant dark objects bluish.

We can divide the scattering phenomena into two groups: those that remove light from a ray (absorption and out-scattering, which we will combine under the term "extinction"), and those that add it (in-scattering). Let's compare the radiance in a ray before and after it is affected by atmospheric light scattering (L_0 and $L_{scattering}$, respectively). Extinction has a multiplicative effect on L_0, which we can express as a dimensionless factor F_{ex}. In-scattering has an additive effect on L_0, which we can express as a radiance value L_{in}. This gives us Equation 1:

$$L_{scattering} = F_{ex} L_0 + L_{in} \qquad \text{Eq. 1}$$

The rest of this article will focus on how to calculate F_{ex} and L_{in} for all objects in a scene in real time.

Absorption

The absorption cross section σ_{ab} measures how well a single particle absorbs light around it. In Figure 5 we see a single absorbent particle. The first assumption we will make is that the

particle interacts with light in an isotropic manner, that is, it doesn't matter from which direction the light comes. Given this, we will look only at the light coming from a single direction and ignore light coming from other directions. In this example, the light from this direction has a constant radiance L. The particle will absorb a certain amount of total radiant flux Φ_{ab}, however we only care about the absorbed flux coming from one direction (the absorbed radiant intensity, I_{ab}).

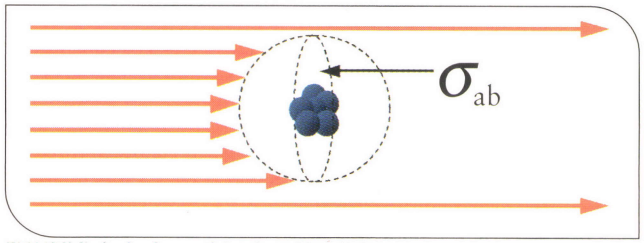


FIGURE 5. A single particle absorbing light.

We define the absorption cross section as the absorbed radiant intensity per unit incident radiance, or I_{ab}/L (this is equivalent to the commonly used definition of absorbed flux per unit irradiance and is easier to explain). If we assume that the particle is a solid absorbent sphere, then for the purpose of absorbing light from this direction we can treat it as a flat disc perpendicular to the light. Each point in this disc absorbs an amount L of radiance. Integrating over the disc's area A gives us $I_{ab} = AL$, so $\sigma_{ab} = A$.

If the particle is very large compared to the light wavelength, then its absorption cross section is equal to its geometric cross section. Smaller particles cannot really be treated as spheres (or as having any shape at all), but fortunately we don't need to care — σ_{ab} captures everything we need to know about how well they absorb light. Note that σ_{ab} varies as a function of wavelength, so it is actually an RGB triple: σ_{ab}^{R}, σ_{ab}^{G}, σ_{ab}^{B}.

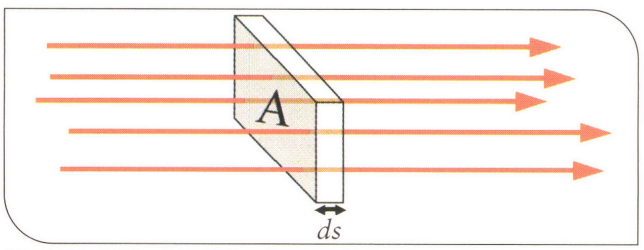


FIGURE 6. A thin slab of absorbent medium.

Understanding how a single particle absorbs light is all well and good, but how is light affected by passing through an absorbent medium containing many such particles? We will characterize this with a new quantity: the medium's absorption coefficient β_{ab}, defined as the particle density ρ_{ab} multiplied by σ_{ab}. Since density is measured in meters^{-3} (particles per cubic meter), the units of β_{ab} work out to be meters^{-1}, or inverse length. This seems a bit odd at first but will make perfect sense in a moment.

In Figure 6, we see a thin slab of absorbent medium with depth ds and area A, through which photons are passing in a perpendicular direction. The total absorption cross-section of the slab is σ_{ab} multiplied by the number of particles in the slab. The number of particles is equal to ρ_{ab} multiplied by the slab volume, which is equal to Ads. This gives us a total absorption area of $A_{ab} = \sigma_{ab}\rho_{ab}Ads$ for the slab. The probability P_{ab} that any given photon will be absorbed is equal to the ratio of the total absorption area to the slab area, which is:

$$P_{ab} = A_{ab}/A = \sigma_{ab}\rho_{ab}ds = \beta_{ab}ds \qquad \text{Eq. 2}$$

So the significance of β_{ab} is that it relates the distance a photon travels through the medium to its chance of being absorbed. This explains why it has units of inverse length — β_{ab} times distance equals probability, a dimensionless number. Another way to look at β_{ab} is that it relates the distance a ray of light travels through the medium to the degree by which its radiance is attenuated by absorption. With this in mind, we can rewrite Equation 2 as a differential equation:

$$\frac{dL}{ds} = -\beta_{ab}L$$

We will assume that β_{ab} is constant along the ray's path. Then we can solve this equation to get the radiance of a ray (with starting radiance L_0) after traveling a distance s through the medium:

$$L(s) = L_0 e^{-\beta_{ab}s}$$

This is a simple exponential decay formula. If β_{ab} is not constant, the solution is more complicated (see Hoffman and Preetham in For More Information). Note that if we have different types of absorbent particles in the medium we can just add their absorption coefficients together and use the sum as the total absorption coefficient. $L(s)$, L_0, and β_{ab} are RGB triples.

Out-Scattering

The derivation for out-scattering is similar to that for absorption. We have a scattering cross-section σ_{sc} (see Figure 7), scattering coefficient $\beta_{sc} = \sigma_{sc}\rho_{sc}$, and the equation for radiance attenuation due to out-scattering in a constant medium is:

$$L(s) = L_0 e^{-\beta_{sc}s}$$

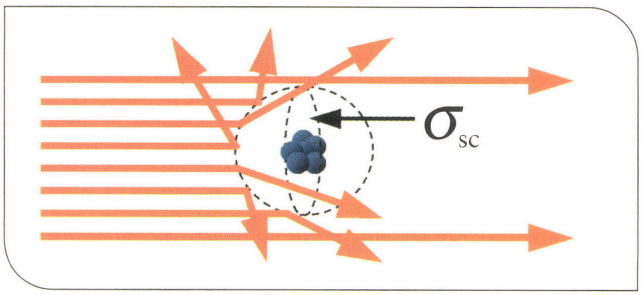


FIGURE 7. A single particle scattering light.

We can also add up the scattering coefficients for different particle types and use the sum as the total scattering coefficient.

Extinction

Since both absorption and out-scattering cause attenuation of light, we can sum the absorption and scattering coefficients to get the extinction coefficient:

$$\beta_{ex} = \beta_{ab} + \beta_{sc}$$

Then the total attenuation due to extinction is:

$$F_{ex}(s) = e^{-\beta_{ex} s} \qquad \text{Eq. 3}$$

In-Scattering

Light is scattered into the view ray from all directions; we will only handle in-scattering from the sun. The scattering coefficient tells us how much light is scattered but not in which direction. For this we define the scattering phase function $f(\theta,\varphi)$. This is a density function for the probability of a photon being scattered in the direction θ,φ. We assume that $f(\theta,\varphi)$ depends only on the angle θ between the incoming direction and the scatter direction ($f(\theta,\varphi) = f(\theta)$), and that it is not wavelength-dependent. Both assumptions are reasonably accurate for most classes of atmospheric particles. The phase function's units are inverse solid angle, and integrating it over the sphere yields a result of 1.0.

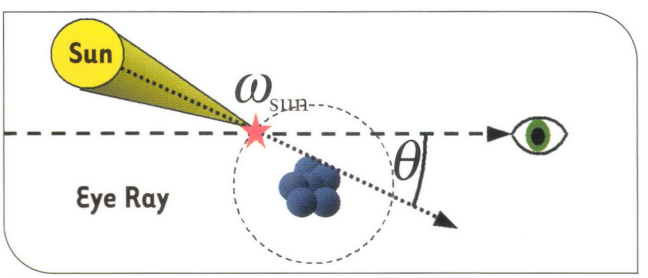


FIGURE 8. A single in-scattering event.

In Figure 8 we can see that the view ray intersects the scattering cross section of the particle, so an in-scattering event is happening (as well as an out-scattering event, but that isn't relevant to the present discussion). How much radiance is scattered into the view ray by this event? First, we need to integrate $f(\theta)$ over the sun's solid angle to get the probability that radiance is in-scattered from there. Since the sun covers a small cone (about half a degree across), we can assume $f(\theta)$ does not vary within it. In this case the in-scattering probability is equal to $f(\theta)\omega_{sun}$. To find the amount of radiance in-scattered by this event, we multiply the in-scattering probability by the sun's radiance, L_{sun}, to get $f(\theta)\omega_{sun}L_{sun}$. We define a new constant, $E_{sun} = \omega_{sun}L_{sun}$, which expresses the total illumination intensity of the sun and is similar to intensity values used in point light source lighting equations (there the intensity is "squeezed" into a zero solid angle, which implies an infinite radiance — point light sources don't really exist). Then the radiance added by a single in-scattering event is $E_{sun}f(\theta)$.

To get the total in-scattered radiance over a short distance ds, we need to multiply $E_{sun}f(\theta)$ by the probability of an in-scattering event, which is $\beta_{sc}ds$. The result is $E_{sun}f(\theta)\beta_{sc}ds$. We define the angular scattering coefficient, $\beta_{sc}(\theta)$, as equal to $\beta_{sc}f(\theta)$. Then the in-scattered radiance over the distance ds is $E_{sun}\beta_{sc}(\theta)ds$. This gives us another differential equation:

$$\frac{dL}{ds} = E_{sun}\beta_{sc}(\theta)$$

Unfortunately, we can't solve this equation without taking extinction into account, since in-scattered light undergoes extinction before it reaches the eye. Adding extinction gives us the following differential equation:

$$\frac{dL}{ds} = E_{sun}\beta_{sc}(\theta) - \beta_{ex}L$$

If we assume E_{sun}, $\beta_{sc}(\theta)$, and β_{ex} are constant along the path, then the solution to this equation is fairly simple (otherwise the solution is much more involved; see Hoffman and Preetham in For More Information). It is essentially Equation 3, plus a new in-scattering factor:

$$L_{in}(s,\theta) = \frac{1}{\beta_{ex}}E_{sun}\beta_{sc}(\theta)\left(1 - e^{-\beta_{ex}s}\right) \qquad \text{Eq. 4}$$

The in-scattered radiance is a function of s (the distance from the eye) and θ (the angle between the viewing ray and the sun). Equations 1, 3, and 4 together describe the complete scattering equation.

Filling in the Parameters

Now that we have the complete scattering equation, we need to determine the parameter values to plug into it: β_{ex}^R, β_{ex}^G, β_{ex}^B, β_{sc}^R, β_{sc}^G, β_{sc}^B, E_{sun}, and $f(\theta)$. E_{sun} is itself dependent on extinction — we will take care of it in the implementation section. In the next two sections we will look at two kinds of particles and determine the coefficients and phase functions for each. We will sum β_{ex} for the two types to get the total β_{ex}, and the two $\beta_{sc}(\theta)$ functions will be added to get the total $\beta_{sc}(\theta)$.

Air Molecules and Rayleigh Scattering

First we will look at particles much smaller than the wavelength of visible light, such as air molecules. These particles do not absorb light, so we will look only at scattering. The scattering coefficients for these particles were discovered by Lord Rayleigh around 1870, so this type of scattering is called Rayleigh scattering (see For More Information). For air we use the following scattering coefficient:

$$\beta_{scAir} = \frac{8\pi^3\left(n^2-1\right)^2}{3N\lambda^4}\left(\frac{6+3p_n}{6-7p_n}\right)$$

Where n is the refractive index of air (a dimensionless quantity, equal to 1.0003 in the visible spectrum), N is the number of molecules per cubic meter (equal to 2.545×1025 for air at 0° C and 1 atmosphere), and p_n is the depolarization factor (a dimensionless quantity, equal to 0.035 for air). Plugging in the values for air, together with the R, G, and B sample frequencies (650, 570, and 475 nm respectively) yields the following numbers:

$$\beta^{R}_{scAir} = 6.95\times10^{-6}m^{-1}$$

$$\beta^{G}_{scAir} = 1.18\times10^{-5}m^{-1}$$

$$\beta^{B}_{scAir} = 2.44\times10^{-5}m^{-1}$$

If your game has significantly different conditions (for example, high altitudes or a planet with very high air pressure), you can work out new values. The important thing to note here is that Rayleigh scattering has a very strong preference for shorter wavelengths, so blue light is scattered much more than red. The Rayleigh phase function we use for air scattering is:

$$f_{Air}(\theta) = \frac{3}{16\pi}\left(1+\cos^2\theta\right)$$

Figure 9 is the polar plot for this function. We can see that Rayleigh scattering is weakly directional and includes equal amounts of forward and backward scattering.

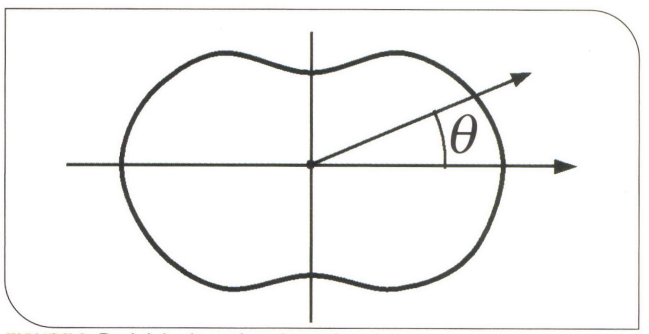


FIGURE 9. Rayleigh phase function polar plot.

Haze Particles and Mie Scattering

Particles much larger than air molecules (soot, dust, water vapor, ice crystals, and so on) are called haze particles. The β_{abHaze} coefficient can vary from 0 to about 5×10^{-5} m^{-1}; it is usually negligible unless there is a lot of pollution present (β_{abHaze} usually has no strong wavelength dependence).

A theoretical model which covers scattering for these particles was published by Gustav Mie in 1908, so this type of scattering is called Mie scattering (see For More Information). Mie equations are very complex and highly dependent on particle size. Haze particle size distributions in the real world are also highly varied, and it is difficult to model β_{scHaze} and $f_{haze}(\theta)$ analytically. Fortunately, many empirical measurements are available. The phase function can be approximated by the Henyey-Greenstein phase function (see For More Information):

$$f_{HG}(\theta) = \frac{(1-g)^2}{4\pi\left(1+g^2-2g\cos(\theta)\right)^{3/2}}$$

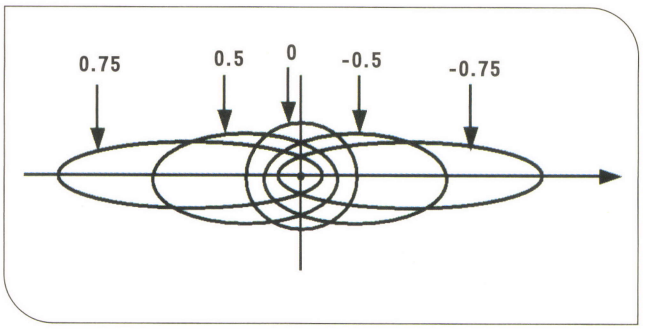


FIGURE 10. Henyey-Greenstein phase function polar plot.

The equation may look scary, but from Figure 10 we can see that this is simply the polar form of an ellipse, where g is the eccentricity parameter (and also controls whether the ellipse points forward or backward). For most haze distributions, g should be negative. As β_{scHaze} increases, g increases in magnitude and β_{scHaze} becomes more monochromatic. A group of typical values derived from empirical measurements can be seen in Table 1.

TABLE 1. Typical haze parameter values.

Description	β^{R}_{scHaze}	β^{G}_{scHaze}	β^{B}_{scHaze}	g
Light haze	2×10^{-5}	3×10^{-5}	4×10^{-5}	-1
Heavy haze	8×10^{-5}	10^{-4}	1.2×10^{-4}	-3
Light fog	9×10^{-4}	10^{-3}	1.1×10^{-3}	-10
Heavy fog	10^{-2}	10^{-2}	10^{-2}	-30

These values are for "normal" real-world environments. For more unusual environments, almost any values can be used; feel free to have a strongly colored absorption coefficient, or even a red-colored scattering coefficient (those even happen in the real world on rare occasions, thus the expression "once in a blue moon").

Aerial Perspective

Aerial perspective is caused by both extinction and in-scattering. Since the viewing rays are close to the ground, the constant density atmospheric model is a reasonable assumption and all the equations hold up. We treat the original (without scattering) color of the object as L_0 and multiply with $F_{ex}(s)$ and add $L_{in}(s,\theta)$ to get the final color. These factors can be precalculated into textures and rendered using functions of s and θ as texture coordinates, calculated per-vertex, or calculated per-pixel either on the fly or in a postprocessing pass. In our implementation we chose to calculate them per-vertex in a vertex shader or vertex program.

This approach makes good use of modern hardware capabilities, enables changing parameters on-the-fly efficiently, and should work reasonably well even on older hardware with software vertex processing. Our implementation happens to use a DirectX 8 pixel shader for combining the factors with the origi-

nal color, but advanced fragment processing is not necessarily required — depending on what else is happening in that pass, it could be possible simply to store the factors in the diffuse and specular vertex colors and combine them using the standard fragment pipeline. We can see L_0, $F_{ex}(s)$, and $L_{in}(s,\theta)$ being combined in Figure 11.

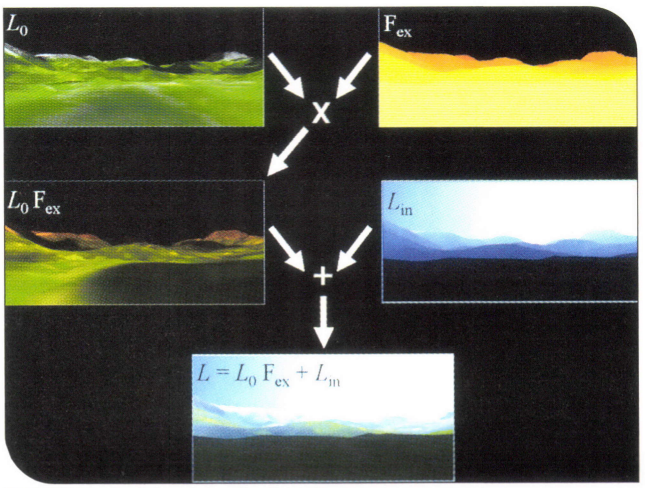


FIGURE 11. Aerial perspective in action.

Sunlight

The sun intensity factor E_{sun} is used for lighting the scene and for in-scattering calculations. We calculate it once a frame by applying an extinction factor $F_{ex}(s)$ to the sun's intensity in outer space, E^0_{sun}. We could get an exact value for E^0_{sun} in watts per meter squared for R, G, and B, but we would still need to convert the resulting radiance values at the end to pixel values we can handle. Since this factor scales every radiance value in the scene, we will simply set it to the largest illumination value we can handle. On older systems this may be 1,1,1 (perhaps 2,2,2 with some careful use of overbrightening techniques), but on newer hardware and graphics engines we should be able to use larger values.

We will use the same parameters and model to calculate $F_{ex}(s)$ as we used for aerial perspective. However, it is not clear what to use for s, and the atmospheric density along the path is far from constant. We solve both problems at the same time by using optical length for s. Optical length is a distance defined as the integrated density along the ray divided by the density at ground level. So if we use our constant-density atmosphere model and use optical length for s, we will get the right extinction results.

The optical length of the atmosphere for air molecules is about 8.4 kilometers at the zenith (straight up). The exact optical length for haze particles depends on various factors, but a reasonable value to use is 1.25 kilometers (haze particles thin out faster with height than air molecules, so the optical length is shorter for them). For other directions, the length follows Equation 5 (see Iqbal in For More Information):

$$l(\theta_s) = \frac{l_{Zenith}}{\cos(\theta_s) + 0.15(93.885 - \theta_s)^{-1.253}} \qquad \text{Eq. 5}$$

Note that in this equation θ_s is in degrees. Since we are using two different values of s (for air molecules and for haze), the equation for F_{ex} looks a little different:

$$F_{ex}(s_{Air}, s_{Haze}) = e^{-(\beta_{exAir} s_{Air} + \beta_{exHaze} s_{Haze})}$$

Sky Color

The sky color in all directions is the result of in-scattering. An accurate model would take multiple scattering into account and would be quite complex and expensive to evaluate, especially since it needs to be evaluated for many points every frame. In this case we go for consistency over accuracy and use the same scattering model for the sky as we used for the sun and other objects. A sky mesh is created, reasonably well tessellated, which conforms in size to the air molecule optical length values in all directions. Note that the sky mesh is always centered on the camera.

The sky mesh needs to be rendered with a similar vertex shader or vertex program to that used for the aerial perspective. The main difference is that here we have different values of s for the two particle types. Fortunately, the ratio between the two is a constant, so we can size the mesh to the air molecule optical lengths and then pass in the ratio as an additional parameter to the vertex shader. The vertex shader can then internally generate s_{Haze} from the vertex distance and the ratio constant. We assume that $s_{Air} > s_{Haze}$, so we can treat the atmosphere as two shells: the inner shell contains both air and haze and the outer contains only air. The resulting equation for L_{in} is:

$$L_{in}(s_{Air}, s_{Haze}, \theta)$$

$$= E_{sun}\left(\left(\frac{\beta_{scAir}(\theta) + \beta_{scHaze}(\theta)}{\beta_{exAir} + \beta_{exHaze}}\right)\left(1 - e^{-(\beta_{exAir} + \beta_{exHaze}) s_{Haze}}\right)\right.$$

$$\left. + \frac{\beta_{scAir}(\theta)}{\beta_{exAir}}\left(1 - e^{-\beta_{exAir}(s_{Air} - s_{Haze})}\right) e^{-(\beta_{exAir} + \beta_{exHaze}) s_{Haze}}\right)$$

Since the sky uses a different vertex shader, we can also take advantage of the fact that the starting color is black (outer space) and skip the extinction factor calculations. We can also use a simpler fragment pipeline setup that just copies the interpolated in-scattered color.

Vertex Shader

In our implementation, we used a Direct3D 8.1 vertex shader. It should be straightforward to implement this in OpenGL as well, given access to the appropriate extensions. The vertex shader computes $F_{ex}(s)$, and $L_{in}(s,\theta)$, then writes them into `oD0` and `oD1`.

The inputs to the vertex shader are vertex position, transformation matrices, sunlight intensity factor E_{sun}, the sun direction (for computing $\cos\theta$), the various extinction and scattering coefficients, and the Henyey-Greenstein asymmetry factor g. The equations to compute $F_{ex}(s)$ and $L_{in}(s,\theta)$ are the same ones presented earlier in this article, and the vertex shader is a straightforward implementation of these equations. Our current version (which has not yet been thoroughly optimized) uses 33 instructions (not including macro expansions) and eight temporary registers.

FIGURES 12A–12C. Results of low haze with Rayleigh scattering (12a), high haze with Mie scattering (12b), and intermediate haze with a low sun angle (12c).

Results and Sample Demo

We can see some results in Figures 12a–c. Figure 12a shows a scene with a low concentration of haze particles, so Rayleigh scattering is predominant; the sun is high in the sky. Figure 12b shows a scene with a high concentration of haze particles (Mie scattering is predominant) and a high sun angle. In Figure 12c there is an intermediate haze concentration, and the sun angle is low. These images were rendered on a 600MHz Pentium III with an ATI Radeon 8500 at about 60 fps.

The sample demo (available at www.gdmag.com) requires graphics hardware that supports Direct3D pixel and vertex shaders and includes shader source code. The application has some sliders which control the various parameters and a fly-through demo mode.

Getting Light Right

With the right simplifications and assumptions, a full model of the interaction of light with the atmosphere can be expressed with a few reasonably simple equations. These equations can be evaluated in real time to add light scattering and absorption to any outdoor scene without unduly impacting performance. For future work we would like to make the sky color model more accurate, and also handle clouds within a physical scattering framework.

ACKNOWLEDGEMENTS

We would like to thank Kenny Mitchell for providing the terrain-rendering and lighting engine used as the basis for the demo, and Solomon Srinivasan for help with the fly-through mode.

FOR MORE INFORMATION

Blinn, J. F. "Light Reflection Functions for Simulation of Clouds and Dusty Surfaces." *Computer Graphics*, 16(3): 21–29, July 1982.

Henyey, L. G., and J. L. Greenstein. Diffuse Reflection in the Galaxy. *Astrophysical Journal*, vol. 93: 70, 1941.

Hoffman, N., and A. J. Preetham. "Rendering Outdoor Light Scattering in Real Time." *Proceedings of the Game Developers Conference*, March 2002.

Iqbal, M. An Introduction to Solar Radiation. Academic Press, 1983.

Klassen, R. V. "Modeling the Effect of the Atmosphere on Light." *ACM Transactions on Graphics*, 6(3): 215–237, July 1987.

Mie, G. "Bietage zur Optik truber Medien Speziell Kolloidaler Metallosungen." *Annallen der Physik*, 25(3): 377, 1908.

Preetham, A. J., P. Shirley, and B. E. Smits. "A Practical Analytic Model for Daylight." *Computer Graphics (Proceedings of SIGGRAPH 1999)*: 91–100, August 1999.

Strutt, J. W. (Lord Rayleigh). "On the Light from the Sky, Its Polarization and Colour." *Philosophical Magazine*, vol. 41: 107–120, 274–279, April 1871.

8 years of GameDeveloper for just $189

Each CD-ROM contains:

- ✓ Full text of feature articles, columns, and product reviews
- ✓ Articles, artwork, and diagrams exactly as they were printed
- ✓ All code and applications that accompanied the articles

Great content. Great prices!

NEW!

✓ May 2001 to April 2002 *Ships after May 15, 2002*	*$49*
✓ May 2000 to April 2001	*$39*
✓ May 1999 to April 2000	*$39*
✓ 1994 to April 1999	*$99*

Or get everything:

✓ Premier issue to April 2002	*$189*

Order by phone:
800.444.4881 (US) or 785.841.1631 (international)

Order online:
www.gamasutra.com/store

Sponsored by: Maya™ Alias | wavefront

GamaNetwork

matt maclaurin

Outsourcing Reality

Integrating a Commercial Physics Engine

Licensing rendering engines is now a well-established practice, with great potential cost and time savings over the development of a single game. As game developers reach for new forms of gameplay and a better process for implementing established genres, the wisdom of licensing physics engines is becoming inescapable. Commercial engines such as Havok and Mathengine's Karma (at press time, Criterion Software, makers of the Renderware line of development tools, were in negotiations to acquire Mathengine) have become mature platforms that can save months in development and test. Their robust implementations can provide critical stability from day one, and their advanced features can offer time advantages when developers are exploring new types of gameplay.

This sophistication does come with a cost. Physics engines do more than just knock over boxes, and the interface between your game and a physics engine must be fairly complex in order to harness advanced functionality. Whether you have already licensed an engine and want to maximize your investment or you're just budgeting your next title, gaining a better understanding of the integration process will save a lot of trial and error, and hopefully let you focus on better physics functionality while spending less time watching your avatar sink through the sidewalk.

The bare minimum we expect from a physics engine is fairly obvious: we want to detect when two objects are interacting and we want that interaction to be resolved in a physically realistic way — simple, right? As you progress deep into integration, however, you'll find physics affects your user interface, logic mechanisms, AI routines, player control, and possibly even your rendering pipeline (Figure 1).

MATT MACLAURIN | *Matt writes code for avatars, animation, and physics at Cyan Worlds. In previous lives he spent five years as an engineer at Apple and ran his own studio for eight years. He has a movie special-effects credit and remembers the register names on the 6502, despite years of trying to forget. Contact him at mmaclaurin@hotmail.com.*

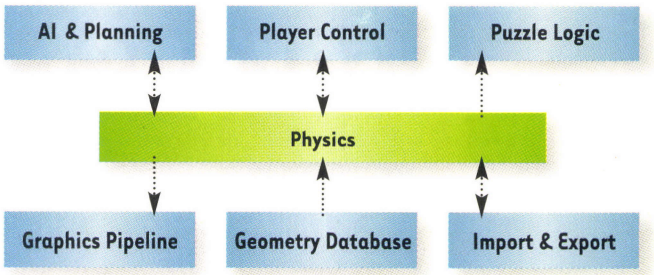


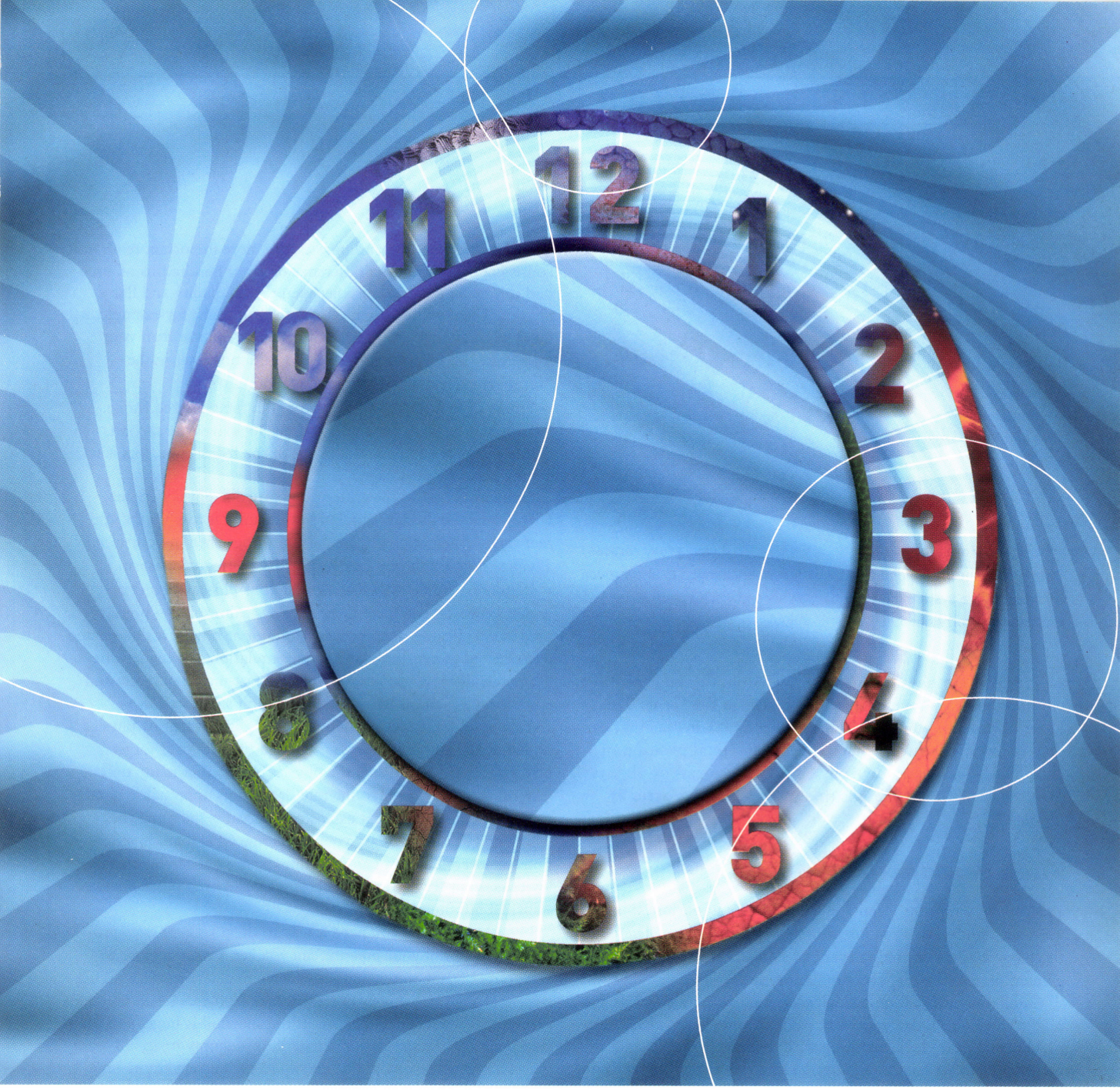
FIGURE 1. Physics has many (inter) faces.

Here at Cyan Worlds, we're more than a year into our use of a commercial physics engine, having integrated it with our own proprietary game engine. I'm going to share with you some of the nuts and bolts of our integration process. In the first part of this article, I'll talk about the fundamentals: data export, time management, spatial queries, and application of forces. Then, with an eye toward character-centric game implementations, I'll visit the twin demons of keyframed motion and player control. In these areas, challenges arise because both of them require that you bend the laws of physics somewhat, and that means you must draw some clear distinctions between what is physics and what is programming for effect.

Integration Basics: Geometry Export

There are three categories of geometry supported by physics engines. The simplest are primitives, represented by formulae such as sphere, plane, cylinder, cube, and capsule. Somewhat more expensive is convex polygonal geometry. Convexity simplifies detection and response greatly, leading to improved performance and better stability. Convex shapes are useful for objects where you need the tighter fit that you can get from a primitive but don't have to have concavity. Finally, there is polygonal geometry of arbitrary complexity, also known as polygon soups. Soups are fairly critical for level geometry such as caves and canyons but are notoriously difficult to implement robustly and must be handled with care to avoid slowdowns.

Since these geometric types have different run-time per-



formance costs, you'll want to make sure that your tools allow artists to choose the cheapest type of physical representation for their artwork. In some cases your engine can automatically build a minimally sized primitive (an implicit proxy) at the artist's request; in other cases the artists must hand-build substitute geometry (an explicit proxy). You'll need to provide a way to link the proxy to the visible geometry it represents, so that changes in the physical state of an object will be visible to the user.

Transforms

Transforms in a rigid-body simulation do not include scale or shear. This mathematical simplification makes them fast and convenient to work with, but it leaves you with the question of what to do with scale on your objects. For static geometry, you can simply prescale the vertices and use an identity matrix. For moving physical geometry, you'll most likely want to forbid scale and shear altogether; there's not much point in having a box that grows and shrinks visually while its physical version stays the same size.

In most cases, a proxy and its visible representation will have the same transform; you want all movement generated from physics to be mirrored exactly in the rendered view. To relieve artists from having to align the transforms manually — and keep error out of your process — you may find it worthwhile to move the vertices from the proxy into the coordinate space of the visible geometry (Figure 2a).

However, if the proxy geometry will be used by several different visible geometries, you may wish to keep the vertices in their original coordinate system and simply swap in the visible geometry's transform (Figure 2b). This method

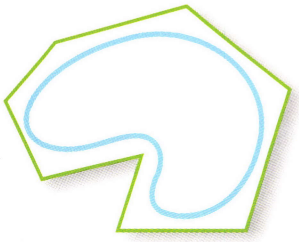


FIGURE 2A. Move proxy vertices (green), into the visible geometry's coordinate system (blue), to ensure exact correlation.

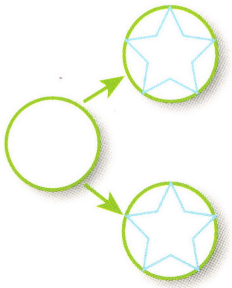


FIGURE 2B. Leave proxy vertices (green) unchanged to allow reuse on several visible geometries (blue).

will let you use physical instances, wherein the same physical body appears several different places in the scene. This latter approach, while enabling efficiency via instancing, can be less intuitive to work with because the final position of the physical geometry depends on the transforms of objects it's used for and not the position in which it was actually modeled.

Time Management

Dealing with time cleanly is an extremely important thing to get right early on in integrating a physics engine. There are three key aspects of time relevant to simulation management: game time, frame time, and simulation time.

Game time is a real-time clock working in seconds. While you might be able to fudge your way from a frame-based clock to a pseudo-real-time clock, working with seconds from the start will give you a strong common language for communicating with the physics subsystems. The more detailed your interactions between game logic, animation, and physics, the more important temporal consistency becomes — a difference of a few hundredths of a second can mean the difference between robust quality and flaky physics. There will be situations where you want, for example, to query your animation system at a higher resolution than your frame rate. I'll talk about this kind of situation later in the "Integrating Keyframed Motion" section.

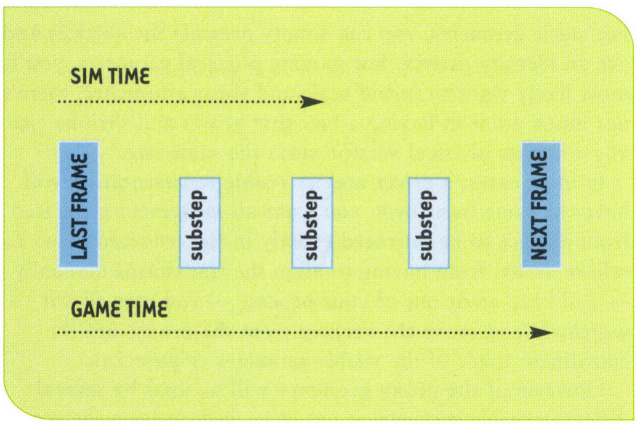


FIGURE 3. The simulation takes smaller steps than the game clock. They meet at frame boundaries.

Frame time is the moment captured in the rendered frame. Picture it as a strobe light going off at 30 frames per second. While you only get an actual image at the frame time, lots is happening between the images.

Simulation time is the current time in your physics engine. Each frame, you'll step simulation time until it reaches the current target frame time (Figure 3). Choosing when in your loop to advance simulation can greatly affect rendering parallelism.

Rendering frame rates can vary; if your physics step size varies, however, you'll see different physical results — objects may miss collisions at some rates and not at others. It's also often necessary to increment, or step, the simulation at a higher rate than your display; physics will manage fast-moving objects and complex interactions more accurately with small step sizes.

Tuning your physics resolution is straightforward. At physics update time, simply divide your elapsed time by your target physics frequency and step the physics engine that many times. Careful though, if your frame rate drops, this approach will take more physics steps so that each step interval is the same size, which will in turn increase your per-frame CPU load. In situations of severe lag, this can steal time from your render cycle, lowering your frame rate, which then causes even more physics steps, ad infinitum.

In such scenarios, you need a way to drop your physics-processing load until your pipeline can recover. If you're close to your target frame rate, you may be able to get away with taking larger substeps, effectively decreasing your physics resolution and accepting a reduction in realism. If the shortfall is huge, you can skip updating the simulation altogether — simply freeze all objects, bring the simulation time up to the current frame time, and then unfreeze the objects. This process will prevent the degeneracies associated with low physics resolution, but you'll have to make sure that systems that interact with physics — such as animation — are similarly suspended for this time segment.

If you're receiving events from the physics engine, the difference in clock resolution between graphics and physics has another implication: for each rendering frame, you'll get several copies, for example, of the same contact event. Since it's unlikely that recipients of these messages — such as scripting logic — are working at physics resolution, you'll need to filter out these redundant messages.

Applying Forces

There are three ways to give an object motion in a physics world: you can apply a force to the object, you can apply an impulse, and you can set its velocity directly. Each has different trade-offs.

To be effective, a force has to be applied over a specific amount of time. In many sims, applying a force means "apply this force over the next simulation step." This is usually not what you want, as applying a force for 1/60th of a second won't push it very far unless it's a huge force. What you do want is a way to say, as simply as possible, "apply this amount of force for this amount of time." There are three ways to do this.

The first approach is to continually reapply the force each substep until you've reached your target time. For each force you wish to apply, keep track of how long it needs to be applied, and apply it one substep at a time. The problem with this approach is its complexity; you need to keep track of each force that you're applying, how long it's been applied for, and how much longer it's going to be applied. There's also the minor problem that you must apply forces over an integer number of substeps, which limits how finely you can tune your use of forces.

What you want is a way to say, as simply as possible, "apply this amount of force for this amount of time."

The second approach is to use impulses. An impulse is a force premultiplied by a time and which takes effect instantaneously. If you want to apply a force of 10 newtons continuously over 1/10th of a second, a 1-newton impulse will do the trick. The limitation to using impulses is that the force is not in fact applied for the entire time; all the energy is delivered instantly, and your object reaches its target velocity instantaneously rather that being gradually accelerated. For quick forces, such as a jump or a bullet, the simplicity of impulses makes them preferable to actual forces. If you want to lift something slowly, though, forces are the way to go.

The third approach — velocities — is both limiting and particularly useful for situations where you need very tight control. We'll discuss it in detail later in the "Player Control Strategies" section.

Spatial Queries

Physics engines by their nature incorporate high-performance spatial data structures. These are handy for a lot of query types:

- **Trigger volumes** (switch to camera B when the user enters this region).
- **Line-of-sight** (can I see the power tower from here?).
- **Ray casts for AI environment probing** (can Watson see me?).
- **Proximity queries for AI** (start talking when the player is within five feet).
- **Evaluating theoretical object placement** (can this door close without crushing anything?).
- **Ray casts for picking** (let the user click on the lever).
- **Volume queries for motion planning** (can I walk all the way to the hatch?).

Spatial queries can affect many types of game logic. A good query interface will save you time every day; it's an area of integration that will reward careful planning. While it can be very game specific, there are a few design parameters for your query interface that apply to almost all games:

Cascading. One query can significantly narrow the field for multiple, more complex queries: a 20-foot sphere around your avatar can gather all potentially interesting objects for subsequent query by line-of-sight.

Triggers. Some queries are set up once and report only when their state changes. For example, a region might notify you when the player enters, rather than you having to ask all regions each frame. This will typically be delivered as an event from the collision system.

Explicit queries. Some queries are only relevant at a particular moment and must be resolved instantaneously, for example, "Is that door in my way?"

Query partitioning. Some questions are only asked about specific types of objects; a camera region may only ever care if an avatar enters it, not a creature or rolling boulder. If your physics engine has an "early out" callback, you can use such application-specific type information to partition the query space, eliminating expensive detailed testing for pairs of objects you know will never interact.

Integrating Keyframed Motion

If you're not using physics for a racing game or flight simulation, you're probably looking for interesting gameplay — big complicated machines, moving platforms, and the like. It's likely that many of these will be lovingly hand-animated by your talented artists. Unfortunately hand animation is not obligated to obey the laws of physics. How do we integrate keyframed motion into a physically based simulation?

The approach I'll discuss here is particular to the Havok API; it happens to be what we're using, and a proper discussion of these details requires a bit of specificity. It should be illuminating regardless of your choice in API, however, as it demonstrates how time, movement, and frame rate can all affect your simulation.

There are two primary issues involved with "physicalizing" keyframed animation:

1. Translate motion from the hierarchical scene graph into the flat physics world.
2. Give the physics engine enough information about the moving object to allow it to interact realistically with other, non-keyframed objects.

We've adopted a few simplifying assumptions for keyframed

motion, which greatly simplify implementation while still capturing the essential functionality.

First, we consider keyframed motion to be nonnegotiable. A keyframed sliding wall can push a character, but a character cannot push a keyframed wall.

Our second assumption is that we do not ask the physics engine to resolve interaction between two keyframed systems. Because these systems are hand-animated and initiated by script, avoiding interdependencies is the level author's domain.

When considering the integration of physics and keyframed animation, we first need to gather the local-to-world transforms of all the keyframed objects, as we'll need them to feed positions and velocities into the simulation. Because physics has no sense of hierarchy, you'll need all your kinetic information in world space. One way to do this is to cache matrices as you traverse your scene graph in preparation for rendering. This process gives you the matrix that you need to match the flat transform structure of physics. Because of the no-negotiating rule for keyframed objects, you can go ahead and submit the keyframed objects to your rendering pipeline as you traverse, as physics will not change those transforms. This helps parallelism, since all static and keyframed geometry can be transmitted to the graphics card before physics even starts.

Keyframed objects participate only partially in the simulation; they are not moved by gravity, and other objects hitting them do not impart forces. They are moved only by keyframe data. For this reason, it is necessary to "freeze" the keyframed objects during the simulation phase in which such forces are calculated and applied.

Keyframed objects are further marked at setup time as zero-order-integration objects. This advises physics that these objects are explicitly positioned and instructs the engine to call back during each integration substep. In this callback, you are responsible for updating the position, orientation, linear velocity, and angular velocity for the keyframed object. This information is critical for determining what happens when, say, your avatar is standing on top of that keyframed elevator. Since the physics engine has no knowledge of the forces at work, it's relying on you to help it fake the results.

To illustrate the importance of getting the velocity right, think about the difference between standing on an elevator that's moving down and one that's moving up. In the down case, a collision between you and the elevator should be resolved by you moving down. In the up case, the exact opposite is desired. The only difference here is velocity, and an incorrect result will embed your player up to the knees in the elevator floor — undesirable by most standards.

The process of calculating velocities is a simple matter of interpolating position and orientation from the animated transforms that you stashed away a few paragraphs back. As an alternate, higher-quality-but-higher-cost approach, you can ask your animation system at each physics substep to interpolate a fresh position for you. This extra bit of work can be expensive, because you have to reinterpolate the motion channel not only for the object in question but also for any parent transforms.

What this gains for you is a greater degree of frame rate independence for keyframed physical objects. To illustrate the problem of frame rate dependence, take a look at Figure 4.

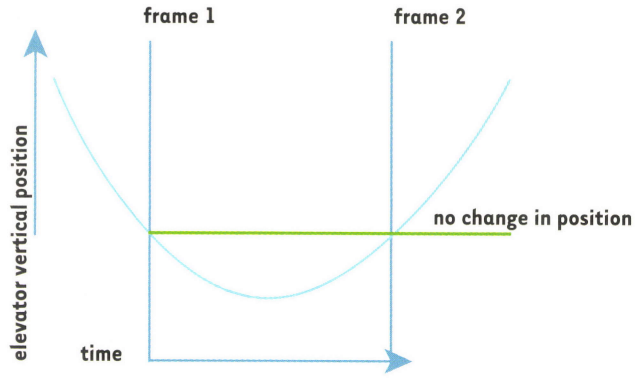


FIGURE 4. Velocity calculation depends on sampling rate.

Figure 4 shows an elevator reaching the bottom of its descent and moving back up. At frames 1 and 2, it's in the same position but moving in two different directions. If you're sampling position only at frame boundaries, you'll conclude that the elevator is stationary. If you add a sample in the middle, you'll have a more accurate simulation, at a cost of reaccumulating all transform dependencies. This is a fairly dramatic case; in many other cases, you'll see the object calculate different velocities at different frame rates. How much this matters to your players depends in large degree on your game's animation speed, object velocities, and tolerance for error in motion. In a surprising number of cases, this winds up not mattering, but it's an accuracy trade-off of which you should be well aware.

The approach I just outlined is not the only one to handling keyframed motion. The Karma engine provides a different facility in which the keyframe data is used as a constraint to the object's position but does not control it directly. The end result is that the object is attached to the animation in a springy fashion; if there are a lot of people in your keyframed elevator, it will lag behind, springing ahead again as folks jump off. You

Big deal, you say, we have a fancy physics package.

can adjust the strength of the spring and the speed with which it acts. This is a neat gameplay effect and can be excellent for the right application.

Player Control Strategies

Player control of the avatar is, for many games, where you're going to spend the most time fine-tuning your physics integration. Every design trade-off you've made regarding physics resolution, applying forces, keyframe data, and the like will all come together to affect how your character navigates and how realistic it feels. The avatar is so central to the player's perceptions that any glitch becomes extremely visible. I'm going to talk about the strategy we're using for our application, a multiplayer, networked, third-person exploration game with a mix of indoor and outdoor environments and an emphasis on photorealism. Naturally, your approach will vary depending on the design of your game, but you'll probably recognize issues that apply to your own situation.

A key decision for player control is the shape of the proxy you'll use to do collision for your character. A popular choice is a simple capsule (Figure 5). This shape has several advantages: It's smooth on the bottom, so it can glide over uneven terrain; it's radially symmetric from above, so your avatar can turn in place without being pushed away from the wall; and it has no sharp corners, which can get caught on narrow doorways. A subtler advantage is that since it presents no sharp corners to the ground, it won't jump or stick as it hits polygon joins in an otherwise flat terrain.

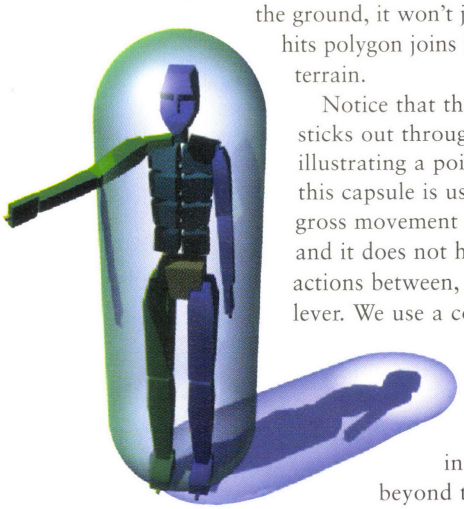


FIGURE 5.
An avatar and his capsule-shaped proxy.

Notice that the character's arm sticks out through the capsule. He's illustrating a point, which is that this capsule is used only for his gross movement in the environment, and it does not handle detail interactions between, say, his hand and a lever. We use a completely different mechanism for such detail interactions; the problems of detail interaction are beyond the scope of this article, but suffice it to say that they're different enough to justify separate mechanisms from those used for movement. As for the realism of the simplistic shape, it's instructive to note that a large percentage of a human's motor control goes into maintaining the illusion that we're not a bundle of flailing limbs all moving in different directions. A real human body does an extremely good job of moving our head along on a smooth path. As a result, a simplified physical body can actually lead to more realistic results than a multi-limbed physics body.

That's how we're shaped, but how do we move? What translates button presses into forward motion? There are three fundamental approaches. First you can set the position and orientation of your character directly. Second you can set the velocity (linear and angular) of your character. And finally, you can apply forces to propel your character.

Setting position is attractive because it's so simple: You're standing here and you want to move forward, so just add a vector. This approach falls apart pretty quickly, unfortunately, and it is the least friendly to using physics in a general fashion.

Assume we start each frame in a physically valid position. Our player tells us to move forward, so we construct a vector representing typical forward motion, orient it to our player's forward vector, and add it to our position. Easy enough so far, and if all games were played on an infinite flat plane, this would work great. But what happens when the position we want to occupy overlaps with a wall, or even with a slight rise in the ground?

Big deal, you say, we have a fancy physics package. We'll just ask it to validate the position before we finalize it. So what do you do when the position is not valid? You'll have to calculate the point of impact, figure out where your character is deflected, and so on. This situation only gets worse when you consider that there are other moving objects in the environment. The problem is that by setting position directly, you've shut your physics engine out of the loop and you now have to write more code to take its place. How do we get physics to do this work for us?

Forces are a natural way to move a physics body around. On the good side, you'll find that a lot of unplanned situations tend to work when you use forces: If your character hits some boxes, he'll knock them over. If he's hit by a rolling boulder, the force imparted by the boulder will combine with his walking force to move him in a new direction. He'll interact realistically with slopes and walls. In general, it's a major improvement.

On the other hand, using forces to move the player somewhat decreases your level of control over exactly how the player moves. Subtler issues such as friction come into play, and it becomes hard simply to say, "Walk to this spot." Forces tend to highlight the fact that we're using a simplistic capsule shape for the player and not a 400-bone muscu-

loskeletal simulation. While a golf ball might fly 100 yards if you whack it with a paddle, a human won't, and the reasons why are a complex to emulate.

Positioning the player by setting velocity is a reasonably happy medium between the total physics-unfriendliness of setting position and the loose control provided by forces. Rather than saying what position you want to be in each frame, calculate how fast you need to be moving to reach your target position and set the velocity on your physics body accordingly.

Physics engines are going to do for gameplay what rendering engines have done for visuals.

This has many of the same benefits as forces. If your character hits a wall, he'll either stop or slide along it. If he steps off a cliff, he'll start to fall, and if he hits a slope he'll climb up it. Little rises and falls in the ground will be automatically incorporated into your character's movement, and you still have pretty tight frame-to-frame control of your character's movement; he won't go flying off down a hill if you're setting his speed each frame, and you won't get an unfortunate confluence of external influences causing him to fly through the air.

One drawback to this approach is that your motion is still based on movement on a flat plane, so you're going to see some unrealistic movement when, for example, the ground drops away rapidly. If you're just applying that forward-walk vector, downward gravitational force will be applied every frame, but it will be blown away by your preordained velocity. As a result, the character will fall at a slow, constant rate and won't accelerate toward the ground as he should; he'll only get one frame's worth of acceleration each time before starting over at zero.

There are two solutions to this problem. The first is to leave vertical velocity alone when you're walking, and the second is to stop walking when you're in the air. In actuality, both are necessary; you don't want a single-frame departure from the ground (common when hitting a bump) to interrupt your forward progress, so your walk behavior should continue for a short time after leaving the ground. Since this can cause a few frames of floating when stepping off a cliff, not setting vertical velocity is necessary to trim off any extra frames of floating when cresting a peak. A rule of thumb is that each navigational state should have a sense of what kind of velocity it can set: a walk can't set vertical velocity, but a jump can.

Another drawback to the velocity-based approach is that it does not automatically integrate external forces. If your avatar is walking forward and suddenly slammed by a 10-ton rolling boulder moving left, he won't budge unless you take extra measures to notice that the velocity you sent down last frame has been modified somewhat. Resolving this correctly is somewhat beyond our scope here, but it involves keeping track of the intended velocity and combining it intelligently with the actual velocity, rather than just setting it.

We've just touched on a few of the issues regarding player control in a physical environment. While they can be extremely challenging, solving these problems creatively will open up a lot of new possibilities.

Focus on Creativity

Now that we've been freed of the burden of writing yet another BSP-versus-bouncing spheres physics engine, we find that integrating a full-featured commercial engine can be just as much work. The critical difference between the two approaches is huge, though: a robust implementation of fully generalized physics is capable of forms of gameplay we haven't even dreamed of yet.

I think that physics engines are going to do for gameplay what rendering engines have done for visuals: provide a rich base of stable features, freeing implementers to focus on creative new functionality rather than being chained to an endless wheel of reinvention. We've already seen our play-testers using the laws of physics to invent new gameplay for which we hadn't even planned. Managed carefully, this combination of planning and discovery holds great promise for the future of games and gameplay.

FOR MORE INFORMATION

Havok
www.havok.com

Mathengine's Karma
www.mathengine.com

Source code for calculating a tight-fitting spherical primitive from a polytope:
http://vision.ucsd.edu/~dwhite/ball.html

Generating convex hulls from arbitrary geometry:
www.geom.umn.edu/software/qhull

Using BSPs to break a level into convex shapes:
www.faqs.org/faqs/graphics/bsptree-faq

UNC's excellent pages on collision, with several academic implementations:
www.cs.unc.edu/~geom/collide/packages.shtml

Russell Smith's excellent open-source physics engine, ODE:
www.q12.org/ode/ode.html

Rewarding Innovation in Independent Games

The 5th Annual Independent Games Festival 2003 Call for Entries

IGF Competition
Submissions due: September 1, 2002

Open to all independent game development teams, the IGF Competition offers over $20,000 in cash prizes to developers.

IGF Student Showcase
Submissions due: November 15, 2002

A non-competitive exhibition of games developed by full-time students, designed to showcase the game development talent coming out of high schools and colleges today.

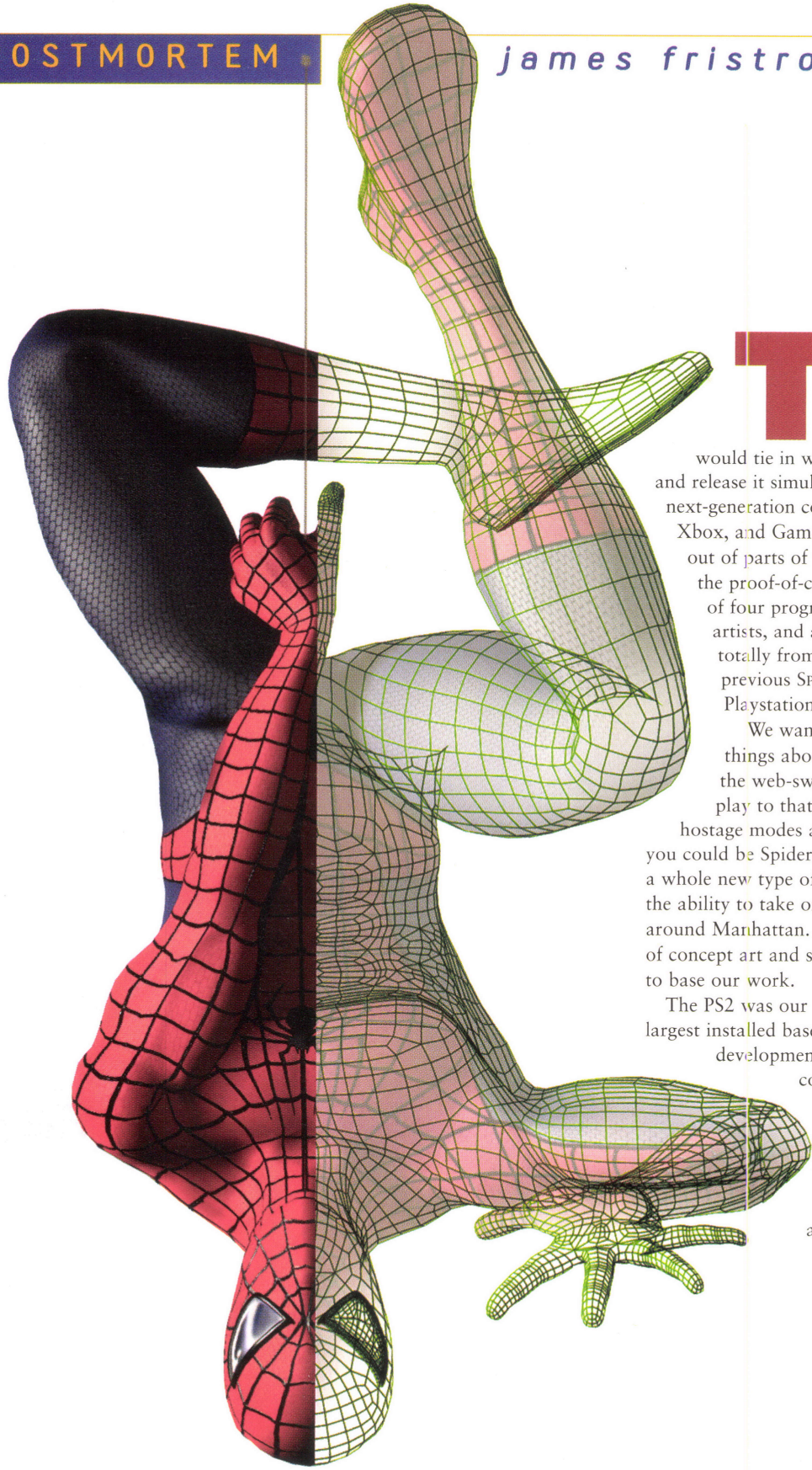
The 5th Annual Independent Games Festival
March 6-8, 2003 · San Jose, California
Entry forms and eligibility rules available at:
www.igf.com

Josh and Dan Goldstein, members of Shizmoo, accepting the Audience Award for KUNG FU CHESS at the 2002 IGF.

INDEPENDENT GAMES FESTIVAL

GamaNetwork

james fristrom



Treyarch was finishing up MAX STEEL and TONY HAWK'S PRO SKATER 2 for Dreamcast when we agreed to do a game that would tie in with the *Spider-Man* movie, and release it simultaneously on all three of the next-generation consoles: Playstation 2 (PS2), Xbox, and Gamecube. We formed a new team out of parts of the others to begin work on the proof-of-concept and design. This team of four programmers, four designers, four artists, and a producer wasn't starting totally from scratch; we had Activision's previous SPIDER-MAN game for the Playstation (PSX) to look at.

We wanted to improve some of the things about the game, such as giving the web-swinging more freedom, and play to that game's strengths, such as the hostage modes and variety of ways in which you could be Spider-Man. We also wanted to add a whole new type of gameplay: aerial combat, the ability to take on flying villains as you swing around Manhattan. Sony/Columbia gave us a ton of concept art and stills from the movie on which to base our work.

The PS2 was our lead SKU, because it has the largest installed base and was the easiest to get development kits for. We figured if we could make our game run on the PS2, we could make it run on anything. We began work on the Xbox a few months after the PS2 and the Gamecube a few months after that.

JAMES FRISTROM | *James is the lead programmer on the SPIDER-MAN team at Treyarch. He occasionally posts his thoughts on game development at http://fristrom.editthispage.com.*

Postmortem: Treyarch's

SPIDER-MAN

What Went Right

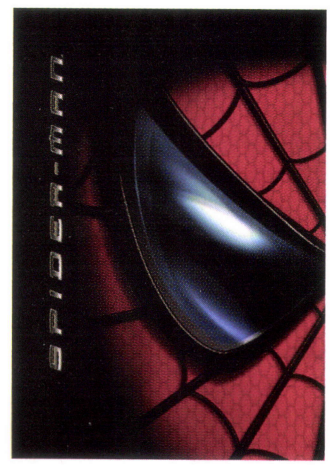
1. Good people. "All you need is good people," says the submarine commander in *Das Boot*, after the chief engineer repairs the boat and saves their lives. It's true, with good people you don't need to enforce process because it happens automatically. Everyone makes sure they do a good job: they volunteer for code reviews, they write their own unit tests, they find better ways to do things instead of the ways passed down from on high.

Our team was made up of a number of talented individuals, who each made their own unique, lifesaving contributions. Even the interns were amazing.

Because I'm not in HR, I don't know how we got these people, but I think part of the reason is that the people in charge of interviewing know their stuff, whether it's art or code. Part of it is the referral bonus we give employees for recommending new hires; part of it is that Treyarch is an environment not too many people are willing to leave. (We've seen what's out there, and it's worse.) Part of it may be the free soda, and free dinners during crunch time, and part of it is our policy of not hiring just anyone to fill a position. (We do hire out of desperation occasionally, but rarely. And when we discover we've hired someone who is just average, we let that person go.)

This team was not only talented but also motivated. Even though adding unplanned features was discouraged, many of us stayed late, on our own time, to get stuff in we thought would make a real improvement to the game. This is where the playing-as-Green-Goblin mode came from, and the secret bowling level, the rats in the sewer level, and a lot of special effects.

2. Developed cross-platform libraries and intermediate file formats. Treyarch was developing a few titles for the next-generation platforms, and it was obvious that having one cross-platform library to do the rendering, which all the teams shared, would be a big advantage. We formed a new team, Next-Generation Libraries (NGL for short). The SPIDER-MAN graphics/PS2 programmer split off from our group to lead the team, providing the architecture and API to which the various platform graphics libraries would be written, and he developed the initial PS2 graphics library.



GAME DATA

PUBLISHER: Activision
FULL-TIME DEVELOPERS: 30
PART-TIME DEVELOPERS: 20
LENGTH OF DEVELOPMENT: 18 months
RELEASE DATE: April 2002
PLATFORMS: Xbox, Gamecube, PS2
DEVELOPMENT HARDWARE USED (average): Programmers used dual Pentium Dell boxes with GeForce 2s, averaging about one and a half development kits per programmer. Artists used single Pentium Dell boxes, and approximately every other artist had a Debug Station with ProView to look at their work.
DEVELOPMENT SOFTWARE USED: PS2: ProDG and ProView; Xbox: Visual C++; Gamecube: Codewarrior. WinCVS for code and SourceSafe for binary, plus 3DS Max 3.1 with Character Studio.
NOTABLE TECHNOLOGIES: NGL, in-house cross-platform libraries.
PROJECT SIZE: Approximately 300,000 lines of C++ code, much of it legacy and obsolete.

NGL had disadvantages as well as advantages: the advantage of more efficient use of coder resources was balanced with the worry that we might not be able to rely on a separate team. The NGL team wasn't always kept in the loop about when our deliverables were due and sometimes weren't available on a weekend or night when they were needed. Finger-pointing would occur: is it an NGL bug or a client-side bug? And sometimes finger-pointing didn't happen when it needed to: a client-side bug would linger on the plate of an NGL programmer who didn't know how to fix it.

At times, the NGL programmers became de facto SPIDER-MAN programmers. They were building the SPIDER-MAN code base on their machines to make sure their changes worked with it, to optimize for the game's worst cases, and to make sure the Xbox and Gamecube matched the PS2 close enough. In the end, NGL worked out great, finishing features and fixing bugs in time to ship.

A still from a movie within the SPIDER-MAN game.

3. Engine reuse. The first decision we had to make was what engine to use. (Writing one completely from scratch was out of the question for an 18-month project, a lesson we learned the hard way with DRACONUS.) We had access to the previous SPIDER-MAN PSX engine through Activision, but our designers and programmers were used to the engine we used for MAX STEEL and DRACONUS, an engine that was already next-generation and cross-platform. Our engine had a powerful scripting language, but it also had slow turnaround times and was never intended for a SPIDER-MAN game.

Despite its shortcomings we felt that this engine was the one to use, and it turned out we were right: we were able to get Spidey swinging through Manhattan on the PC in record time. Now all we had to do was port it to three platforms and add as many features as we could.

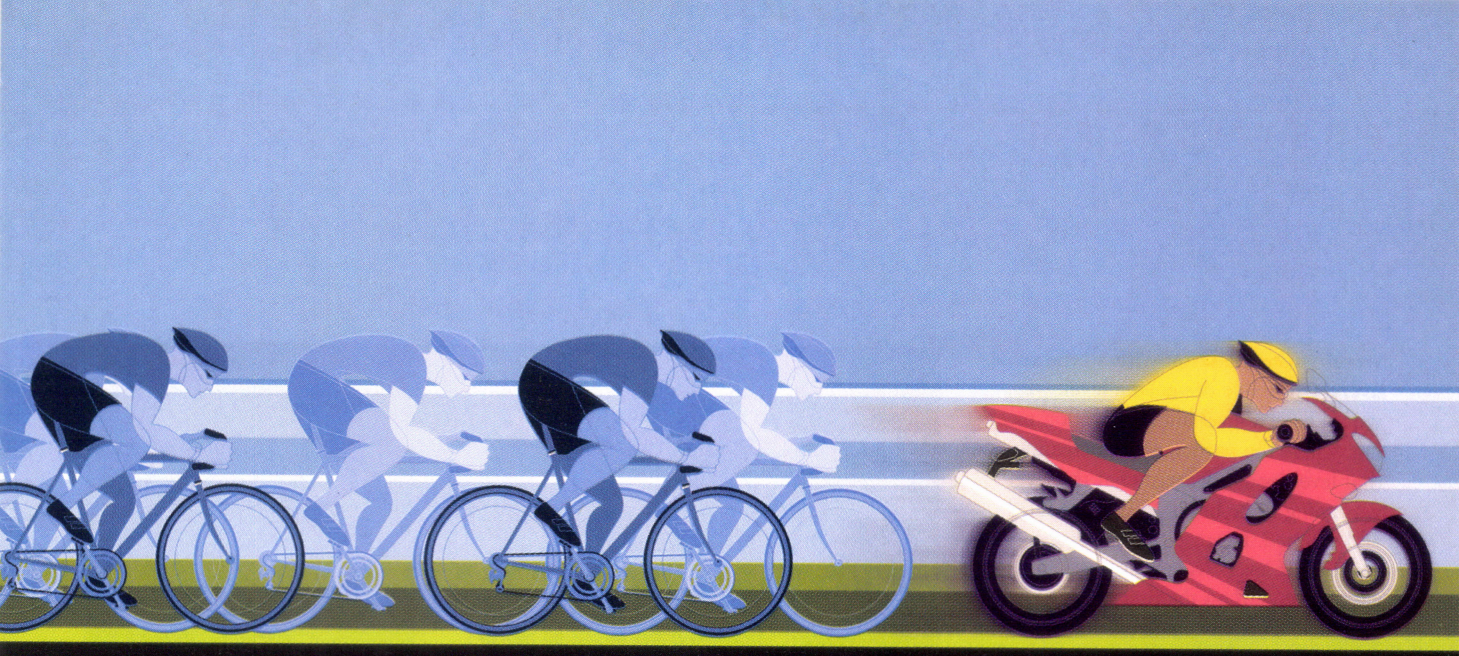
4. Good process. At its heart, our methodology was "code and fix." However, there were many semiformal processes that prevented our software cowboyism from totally spiraling out of control.

The artists and designers were scheduled in Microsoft Project. For the coders we started with an XP-like system of file cards representing weeks, posted up to a large corkboard. But the schedule changed so frequently that this method stopped being good enough, and we switched to Joel Spolsky's method from www.joelonsoftware.com. This worked pretty well; I would go through the schedule every morning to make sure everyone was keeping their schedules up to date. Also, once we switched to this system our estimates tended to be more accurate, which was a nice side bonus.

Using Microsoft Access, we started logging bugs right away. People would log bugs by sending an e-mail to the producer, who would first log the bug into the database and then later print out paper lists for people. We broke bug priorities down into: 0 — don't leave your desk until it's fixed; 1 — fix ASAP; 2 — fix ASAP unless you're holding somebody up; and 3 — this one can slide until the next milestone. Our policy was to fix the bugs first, which meant that most bugs were marked with priorities of 2 or lower. As the project grew, it became clear that the work of maintaining the bug database was sucking down most of our producer's time, and after trying an off-the-shelf bug tracker that didn't meet our needs, we built one in-house using Streamline Technology's Seven Simple Steps, which all of the teams at Treyarch use. It's an Access back end with a web front end that e-mails you when you have a bug. This not only freed up a lot of producer time but also became one of the main tools we used to communicate tasks and bugs to others and to make sure those tasks got done.

On average we broke two levels a day, usually due to simple things like not checking in a new file. To protect ourselves from these errors, we did daily build and smoke tests on the PC and the PS2. We did the daily build on a machine that would do a complete update of the data and source depositories, rebuild and recompile the various intermediate files into their final output, and run an automated test of every level, just to see if they ran. (Generally we've found that 95 percent of the bugs we introduce are of the variety that makes levels stop loading at all.) This way, we'd catch bugs up to a day after they were introduced, instead of discovering the hard way, days or weeks later, that a level nobody had touched in a while didn't work.

On the art and design side of things, we would storyboard a movie or create concept art for a level before we began animating or modeling it. A professional



FREE YOURSELF TO DO THE CODE THAT MATTERS

@ 60 HZ...

- 25 High-Resolution Blended Lit Textured Characters with Shadows, 525 Bones (21 per character, 4 weights per vertex)
- High-Resolution Static Textured Environment
- 3-D Overlay GUI, Stereo Sound, AI, Controllers, Particle Systems, Shadows, Shaders
- Additional Animated Skinned Crowd Characters (based on console)

60 HZ

- One Source
- One Database
- One Intrinsic Alchemy Platform

PS2 Xbox* Gamecube PC**

Try it for yourself!
Download Free ArtistPacks for 3ds max™ and Maya™ at www.intrinsic.com

*Xbox adds 95 extra characters with 760 bones
**Gamecube adds 12 extra characters with 96 bones

INTRINSIC ALCHEMY™

THE GAME DEVELOPER'S ULTIMATE TOOL

sales@intrinsic.com

INTRINSIC GRAPHICS™

WWW.INTRINSIC.COM
650.966.6800

©2002, Intrinsic Graphics, Inc. All Rights Reserved. Intrinsic Alchemy and the Intrinsic Graphics logos are trademarks and/or registered trademarks of Intrinsic Graphics, Inc.

writer wrote the script with all the voice-overs. Before we designed a level, we would have a level implementation meeting, where the participants in creating that level would discuss what the level was going to be before it was modeled and scripted. Because of this process, several of our levels were very close to good-enough-to-ship on the first iteration, although several other levels had to be revisited several times before we signed off on them. We had a concept of "level alpha," which meant the level was basically ready to go in the box except for cutscenes (scripted or animated) and voice-over, and game designers didn't work on the next level until they had their previous one at "level alpha."

Finally, we had a small internal testing department. It's fairly standard in the game industry to wait for a game to reach so-called "alpha" (a completely nebulous term) and then put it into QA. Then QA tests the game out of sight of the developers and submits bug reports. Although we did do the standard external QA with Activision, before the game was at alpha we had internal testing. At first this was just one person, but as we got closer to finishing, the number grew. And once the game hit alpha, Activision sent some of their best and brightest testers over to Treyarch (which was easy, since our offices are one block from theirs) to test the game on-site, so they could demonstrate bugs in person, work with the very latest revisions, do testing on development kits, and ask us when they discovered bugs whether or not the bugs were important. They became part of the team, and I think that because we could meet them and see them face-to-face we accorded them more respect than the faceless testers at Activision. By the end of the project, half of our bugs were caught internally; although there were many duplicates, there were many more bugs that external QA never saw.

5. Communication. The layout of our office was geared to encourage communication between the people who needed to communicate. The leads' offices were fairly central in an L-shaped office suite. The game designers were close to the programmers who supported them, which was key in getting things to happen and encouraged additional lunchtime communication. The programmers who dealt with specific platforms were farther away. NGL was on a different floor, but their lead was in a nearby office, and NGL representatives would share offices with us when it was useful for them to do so.

SPIDER-MAN'S Aerial Combat feature lets players take to the skies and battle airborne villains like the Green Goblin over the streets of Manhattan.

All of our design documents were made available on an intranet web site, along with short documents on how to use the tools and a small FAQ that had some common troubleshooting answers.

Finally, we had a lot of management. Greg John oversaw the entire project. There was a producer watching deliverables. There was a creative director. There was a level-modeling lead. There was an animation lead. There was a game design lead. There was a lead programmer. The engineering group — the programmers who wrote the platform-independent, gameplay-related code — had their own lead. Each console had a lead go-to guy who understood that console best. NGL had a lead programmer and a producer. In all we had about one lead for every five people, and all the leads met once a week.

Although communication was good, it could have been better, particularly between the art and design departments. That's something we'll have to work on for the next project.

What Went Wrong

1. Not enough staff soon enough. Some of our other problems can be traced back to staff size. When the project started, we had about four programmers, four game designers, three modelers, one texture artist, one producer, and one creative director. NGL was only in our imaginations at this point. Treyarch was eagerly looking for more staff, but staff doesn't just come out of thin air. Most important, we didn't have a concept artist, we didn't have a writer, and we had so many people working on the proof-of-concept that we didn't really have someone to come up with a full design.

Later, when we rolled into full production of our inadequate design, things got even worse, because some of our veterans were taken away to work on other projects, without adequate replacement. Eventually we were fully staffed and then some; we had more than 40 people at points during the project, and more than 50 worked on it at one point or

NXN
managing complexity
What do you mean they disappeared?
If data loss, lack of collaboration and project management challenges are plaguing your game production, then consider NXN alienbrain.
NXN alienbrain's powerful combination of asset and production management helps entire teams version control their 2D/3D art and code, collaborate in small and large project settings, and maintain up-to-the-minute information on their complex projects. This allows teams to spend less time putting out production fires and more time focusing on what's really important - creating great content.
Get started on NXN alienbrain with our 10-seat Starter Pack for $9,990.
Take control of your game project
alienbrain
Register now for your evaluation copy at: eval@nxn-software.com.
http://www.nxn-software.com

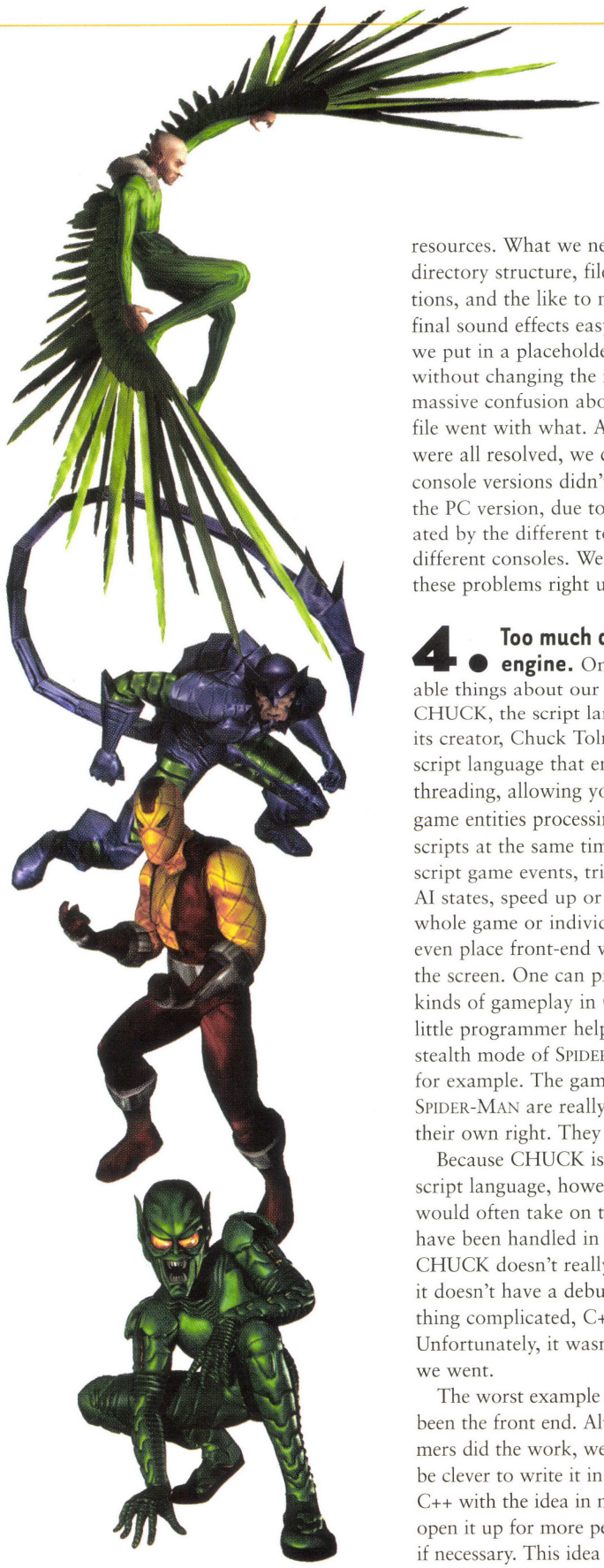
another. We had to make up in the end for what we lacked in the beginning.

2. Inadequate initial design. I think designing is overrated; coming up with a 200-page design document only to scrap most of it seems like a waste of resources. Still, you need some design — a bad plan is better than no plan — and you need more than we had.

By the time we finished our proof-of-concept, the design consisted basically of a list of bosses, a list of levels, and some ideas on how our new AI system might work. Each item had about a paragraph devoted to describing it. Our story was a page long. By the end of the project, we had a wealth of design: we had the output of each level implementation meeting; we had the script; we had a couple of pages on each boss, describing the moves he would be capable of; we had concept art and storyboards; we had concept art for the front end. If we had come up with that material earlier in the project, we would have been rudderless for less time and could have made more game before shipping.

3. Audio and voice-over problems. Because we didn't hire a scriptwriter until fairly late in the project, we didn't get the script approved by Activision and Sony/Columbia until even later. By the time the people who make these decisions had decided to accept the script and get Willem Dafoe and Tobey Maguire to do the voice-over, we were long past the drop-dead date we had given to Activision. It was already alpha, and we had very few of the cutscenes done, because it's a waste of resources to do the cutscenes until you have final voice-over. But we soldiered on. We got the final voice-over recorded and transplanted animators into the SPIDER-MAN animation sweatshop to get all of the cutscenes finished as soon as possible. The end result is that a lot of our cutscenes don't look as good as we would like.

Audio also suffered from massive disorganization. We didn't have anyone on our team dedicated to managing audio resources. What we needed was a good directory structure, file-naming conventions, and the like to make dropping in final sound effects easy. Too many times we put in a placeholder sound effect without changing the name, resulting in massive confusion about which sound file went with what. After those issues were all resolved, we discovered that the console versions didn't sound the same as the PC version, due to discrepancies created by the different tool chains on the different consoles. We were working out these problems right up until we shipped.

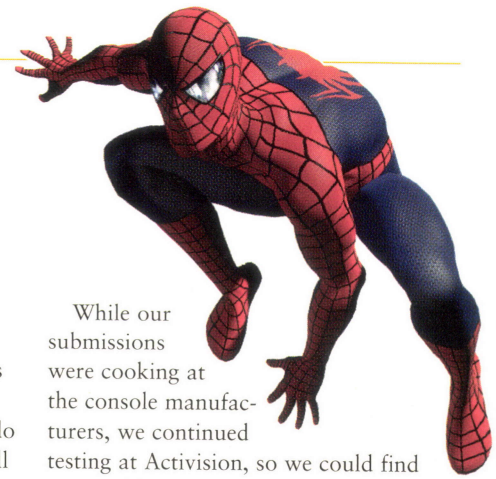


A gallery of villains: Vulture, Scorpion, Shocker, and the Green Goblin

4. Too much done in script engine. One of the most valuable things about our game engine is CHUCK, the script language named after its creator, Chuck Tolman. It's a C++-like script language that emulates multithreading, allowing you to have multiple game entities processing their individual scripts at the same time. It's used to script game events, trigger audio, change AI states, speed up or slow down the whole game or individual entities, and even place front-end widgets and text on the screen. One can prototype whole new kinds of gameplay in CHUCK with very little programmer help; this is where the stealth mode of SPIDER-MAN came from, for example. The game designers on SPIDER-MAN are really programmers in their own right. They are empowered.

Because CHUCK is such a powerful script language, however, game designers would often take on tasks that could have been handled in code better. CHUCK doesn't really have arrays, and it doesn't have a debugger, so for anything complicated, C++ is the way to go. Unfortunately, it wasn't always the way we went.

The worst example of this may have been the front end. Although programmers did the work, we thought it would be clever to write it in CHUCK instead of C++ with the idea in mind that this would open it up for more people to maintain it, if necessary. This idea worked in a way: when we got to the eleventh hour and we still hadn't finished the front end, the game designers stepped in and helped fin-



ish it off. If we had done the front end in code, we would have finished it sooner and the game designers wouldn't have had to step in at all.

Again, this is a result of our staffing difficulties, because if we'd had more staff sooner, programmers could have provided these features in code. Instead, the game designers got fed up and did it themselves, costing us efficiency in the long run.

5. Not enough QA before alpha. Nothing prepared us for how many bugs we'd find in this project. The total came in at around 16,000, about half of which were internal and half of which were external. This was more than twice as large as the largest bug count we'd ever seen on a project at Treyarch before, partly because we were on three platforms and therefore had more bugs. (The same bug on three different platforms might show up as three or even six bugs, as PAL SKUs were being tested independently from their NTSC counterparts.) Still, even though our find rates were high due to the number of platforms, our fix rates were quite ordinary. Our open bug graph looked like Figure 1.

Watching our bug count was like watching the stock market, and we felt lost at sea. On previous projects we'd been able to guess fairly accurately how many bugs we'd have and develop a good idea when we'd be complete. On this project, all we knew was that it was going to take longer than we thought, and possibly a lot longer. All we could do was work extra overtime and mark "will not fix" or "as designed" on as many bugs as we could. At this point in the project, it's a gray area between a "bug" and a feature that "really has to be there." Although we fought against as many changes as we could during this period, most of the battles were lost, as we reluctantly agreed that the opportunity justified the risk of these features. If we'd fought any less hard, or lost any more of those battles against unplanned feature changes, we would not have made our ship date.

Normally we like to have the game in testing for a week after we hit zero bugs, to make sure that there aren't any lingering, hard-to-detect problems. We didn't have that luxury on this project. A week before our final date to submit we still weren't at zero bugs. All we could do was be careful during this week with our fixes: access to SourceSafe was restricted; the team was admonished that they mustn't fix any more low-priority bugs, lest they introduce a stop-shipment bug; and we did informal inspections on all source changes. When we finally hit zero bugs, we worked overnight to get the burns out and submitted the next morning.

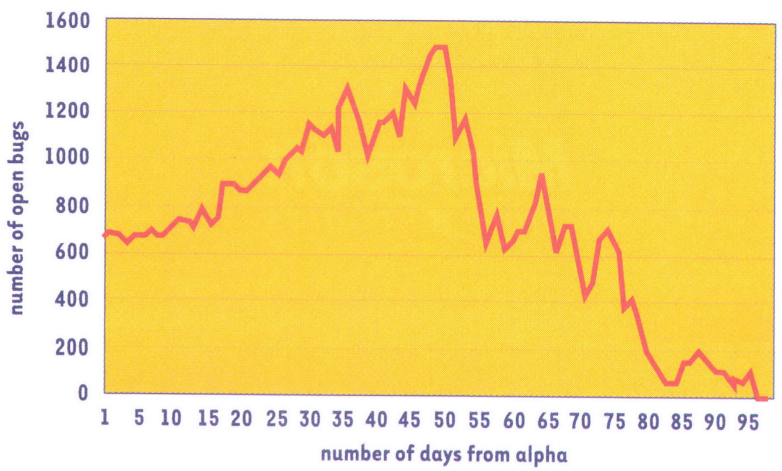


FIGURE 1. SPIDER-MAN's open bug graph.

While our submissions were cooking at the console manufacturers, we continued testing at Activision, so we could find any problems before the console manufacturers did. We found about half a dozen problems that we would have loved to fix, but they were either very rare or not serious enough to warrant a resubmit, although they were quite embarrassing.

We haven't worked out the details of how to reduce our bug count for the next project; more and better and different QA is necessary, but how much and what kind? Formal code reviews? Unit tests? More black-box testing? More soak tests with random monkeys? Maybe we should make asserts and developer-eyes-only error messages fatal, to force people to stop and fix these problems instead of working around them. Maybe we should rely less on the PC version, so we can catch console-specific bugs sooner. Whatever we do, I hope it'll go into the "What Went Right" section of Treyarch's next Postmortem.

One View of Many

This article represents data sifted from the internal postmortem we did at Treyarch. After we shipped the NTSC versions, everyone wrote their lists of what went right and what went wrong on the project. Categorizing that data was difficult, and my lead programmer bias has seeped into what's been presented here. If senior producer Greg John, creative director Chris Soares, or design lead Tomo Moriwaki had written the article, you would have seen a much different view.

Due to the massive movie marketing, the SPIDER-MAN phenomenon is huge. Being a part of it, seeing SPIDER-MAN paraphernalia everywhere, is exciting, and partly makes up for the frustration.

But what really makes up for the frustration is the fact that we somehow pulled it off: an original title, three new platforms, 18 months. And so far, it's doing very well.

ARE YOU EXPERIENCED?

Senior Software Engineer - Graphics

We have an exciting opportunity for a senior graphics developer to join us in pushing the envelope of the Racing and Driving genres for Xbox gaming. Primary responsibilities include designing, implementing, and supporting the graphics pipeline of a key internal title. Candidates should have extensive knowledge on shadowing techniques, lighting and multi-texturing, effects, visibility algorithms, particle systems, and, in general, could be considered an expert in the field. A B.S. or M.S. in Computer Science or a related field and at least three years of relevant experience are required.

Senior Software Engineers - Graphics

We're looking for several talented senior graphics developers for Xbox sports titles. The team will push the Xbox to its technical limits; graphically this means complex lighting effects, bump-mapping, motion blur, and other pixel and vertex shading techniques. Recent Direct3D expertise is required.

Game Designer

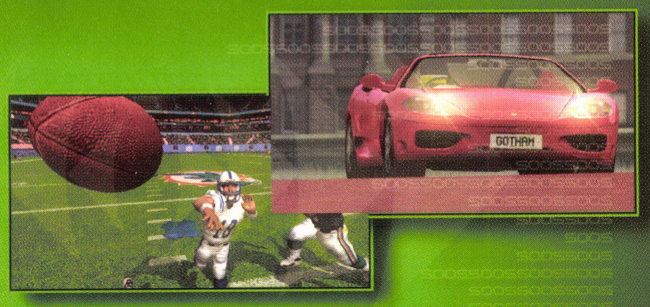
Envision, document, and champion the play and functionality of a game from start to finish including the story, core feature set, game shell, animations, and game flow. Candidates must be able to communicate clearly and concisely with various members of the team via written and verbal means. A minimum of two years experience in a design role at a game company or four years experience in a related industry is required.

Audio Lead

Lend us your ears! The Microsoft Racing Games Studio has an exciting opportunity for a seasoned audio specialist to join us in setting new sonic standards for Xbox and PC racing titles. Responsibilities will include creative audio design, planning, and prototyping, and working closely with internal and external audio providers to guarantee that audio plays a key role in the success of our racing games. 3-5 years of game audio experience and an assertive, positive attitude necessary.

Producer/Program Manager

If sports and sports games are your passion, the Microsoft Sports Studio has a job for you! A strong Producer is needed to drive the development of Team Sports Games for the XBox. Primary responsibilities will be managing all aspects of the game development process, i.e. working with game designers to plan games, writing specifications, relationships with external developers. You will have at least 3 years of experience producing games, or experience in developing retail products; excellent verbal and written communication skills; passion for videogames and sports videogames. Leadership, negotiation, strategic and out-of-the-box thinking a must.



Senior Game Designer

Help us push the envelope in the Xbox and PC Racing/Driving genres. We are looking for Games Industry veterans with vision, passion and consumer focus to join the team behind key titles such as Project Gotham Racing, Midtown Madness, and RalliSport Challenge. Candidates must have a passion for racing/driving games and be prepared to elaborate on the strengths and weaknesses of our products and seek opportunities to innovate. Prior experience shipping leading racing titles is required.

Senior Software Engineer - AI

AI - Just two vowels found in the English language or something much more compelling? We're looking for a developer with a passion to create a cutting-edge AI system for an Xbox sports game. This system will not merely react to the game player's input, it will anticipate and adapt. Position Requirements: Degree in Computer Science or Computer Engineering with a focus on AI; Strong C++ skills; Strong communication skills.

Please visit http://www.microsoft.com/careers.htm to see the full list of available positions.

When submitting your resume to gamejobs@microsoft.com please include on the subject line the job code Aq24u0802. No game concept submissions please.

Microsoft game studios™

© 2002 Microsoft Corporation. All rights reserved. Microsoft is a registered trademark of Microsoft Corporation. Microsoft is an equal opportunity employer and supports workplace diversity.

XBOX

CREATIVE CAREERS

Currently recruiting for key positions in Art, Development, Production, Programming, and other exciting roles.

LucasArts

WE WANT YOU

VISIT
www.lucasarts.com
and click on jobs for details.

GET EDUCATED

DIMENSIONAL ANIMATION i2i

WHERE ANIMATION, LIVE PERFORMANCE & TECHNOLOGY MEET... i2i

- FILMBOX SPECIALIST
- WORLD-CLASS CHARACTER ANIMATION
- TURN MOCAP INTO KNOCK-OUT PERFORMANCE

i2i ANIMATION

www.i2ianimation.com
(323)349-0894

illuminating the digital world

Photoshop Illuminated: For Animators

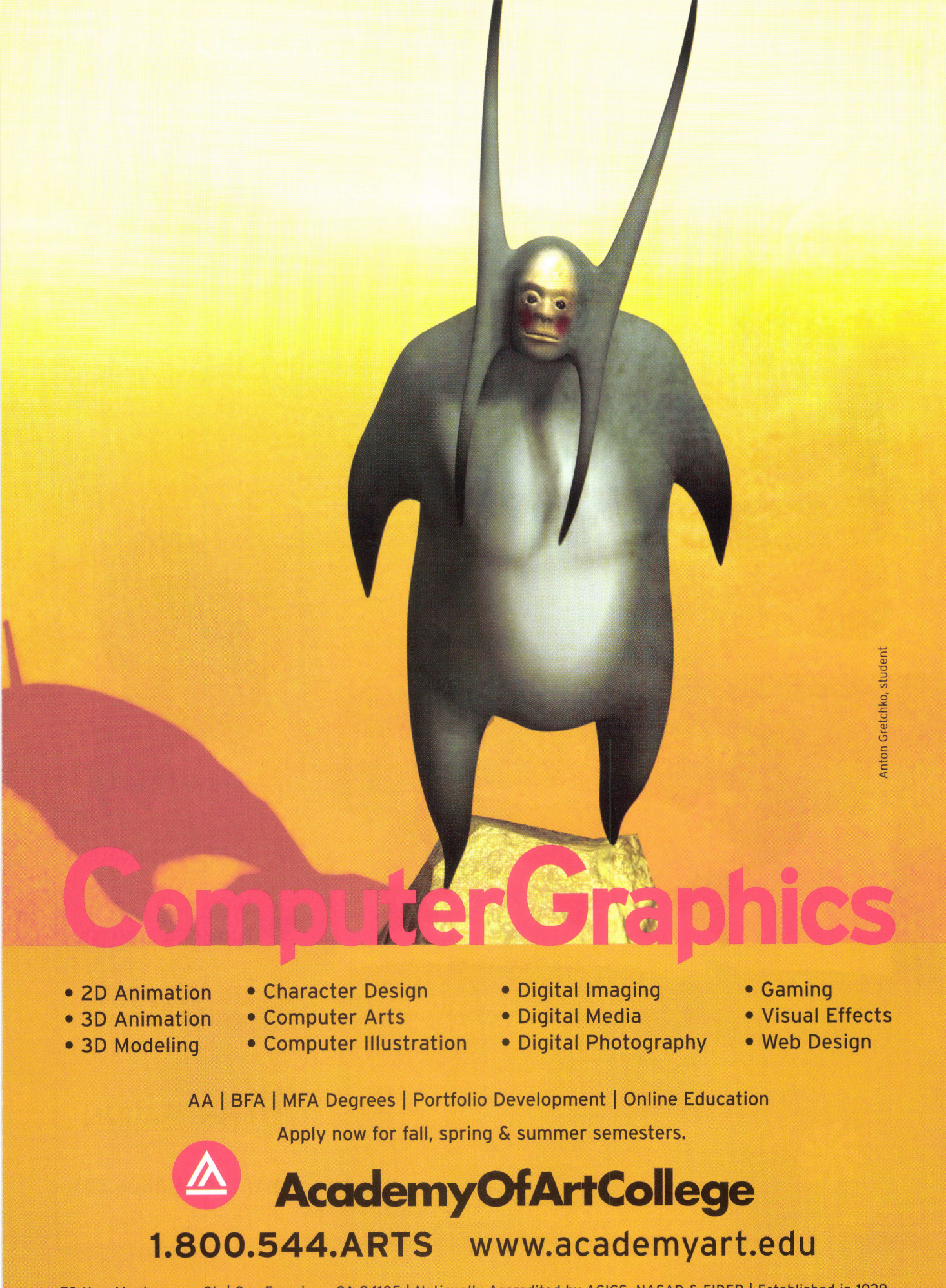
XSI Illuminated: Character

Maya Illuminated: Games

3ds max Illuminated: Foundation

MESMER PRESS

cg books and courseware
for students, teachers, and professionals


Anton Gretchko, student
ComputerGraphics
• 2D Animation
• 3D Animation
• 3D Modeling
• Character Design
• Computer Arts
• Computer Illustration
• Digital Imaging
• Digital Media
• Digital Photography
• Gaming
• Visual Effects
• Web Design
AA | BFA | MFA Degrees | Portfolio Development | Online Education
Apply now for fall, spring & summer semesters.
AcademyOfArtCollege
1.800.544.ARTS www.academyart.edu

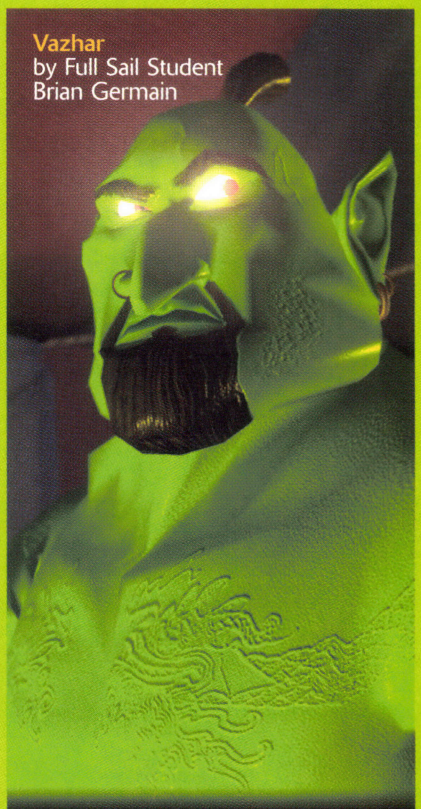
If you're going to pay for an orange...

you should get an orange.

If you squeeze the new DVD from VFS you will never again be confused about who's got the juice.

To get more information or your own DVD of VFS student work call 1-800-661-4101 or email dvd@vfs.com

Vancouver Film School
Creative. Disciplined. Focused.



Vazhar
by Full Sail Student
Brian Germain

FULL SAIL
Real World Education

School of:
Game Design
Computer Animation
Digital Media
Film
Audio
Show Production

800.226.7625
www.fullsail.com

3300 University Boulevard
Winter Park, FL 32792

- Financial aid available to those who qualify.
- Job placement assistance.
- Accredited by ACCSCT.

©2001 Full Sail, Inc. All rights reserved.
The terms "Full Sail," "Full Sail Real World Education," and the Full Sail logo are either registered service marks or service marks of Full Sail, Inc.

UACT

Who says you can't model?
Learn what it takes to be a game designer or game programmer.

Call 1-800-658-5744 | http://www.uact.edu

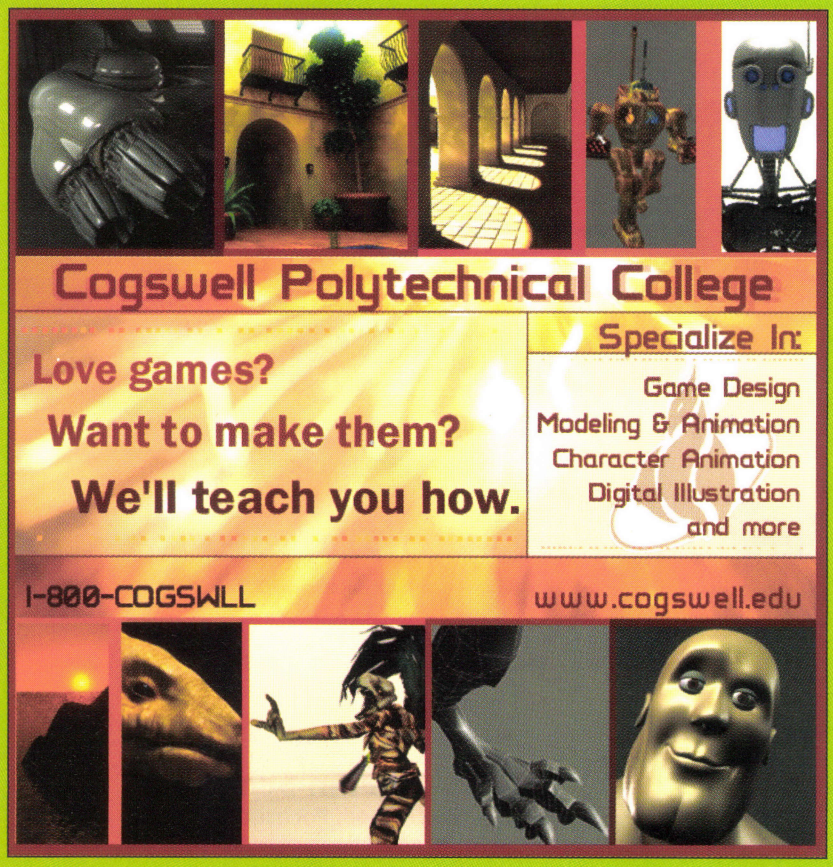
Savannah College of Art and Design

It's Alive!, Alias/Wavefront Maya,™ Adobe Photoshop,® Deep Paint 3D™ and Adobe After Effects®
Terry Ziegelman, St. Louis, Mo., second year M.F.A.;
Paul George, Valencia, Venezuela, second year M.F.A.;
Stephen Johnson, Charleston, S.C., B.F.A.;
Jamie Kirschenbaum, Reston, Va., B.F.A.

BACHELOR OF FINE ARTS
MASTER OF ARCHITECTURE
MASTER OF ARTS
MASTER OF FINE ARTS

Architectural History
Architecture
Art History
Computer Art
Fashion
Fibers
Furniture Design
Graphic Design
Historic Preservation
Illustration
Interior Design
Media and Performing Arts
Metals and Jewelry
Painting
Photography
Product Design
Sequential Art
Video/Film

AN INTERNATIONAL UNIVERSITY FOR THE ARTS
Savannah, Georgia
800.869.7223 | www.scad.edu



we make models
we make textures
we make animations

we made art for these:

THE SIMS
THE SIMS - Livin' Large
THE SIMS - House Party
THE SIMS- Hot Date
THE SIMS - Family Vacation
Myth 3 - The Wolf Age
S3 - Unreal Tournament Edition
...and a lot more

we make art

NEW PENCIL

80 Liberty Ship Way, Suite 6
Sausalito, CA 94965
phone 415.339.1800
fax 415.339.1803
web www.newpencil.com

·MUSIC
·SPEECH ·SOUND
·AUDIO INTEGRATION

OMNI Interactive Audio

- Veteran team with over 10 years game audio experience each.

- Experience on **PlayStation 2**, **Xbox** and **Gamecube**, online **PC** games, mobile platforms.

- Complete audio package, or support for an overworked in-house audio team.

- Hassle free, efficient audio integration including audio tools.

www.OmniInteractiveAudio.com

Let Us Take You to the Next Level.

For years we've helped entertainment and toy companies with licensed product packaging and promotions. Take your project to new heights – put our expertise to work for you.

Packaging • Advertising
Copy writing • Branding
Logo and Icon Development
Marketing Strategy & Messaging
CD / DVD / Video Promos & Demos
Web and Email Campaigns
POP Solutions

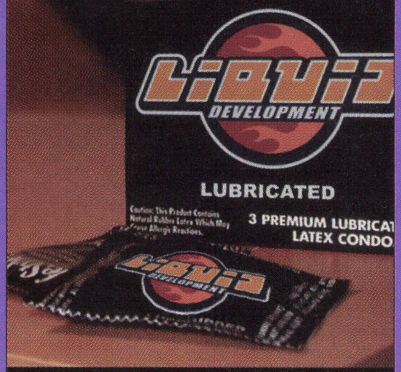
BRAINDANCE

Intelligent creative.
Measurable results.

(323)550-4590
www.braindance.tv
writeus@braindance.tv

If You're Going To Make It...

...Make It Great!



LIQUID DEVELOPMENT

Liquid Development

The Only Art Resource You Need

www.liquiddevelopment.com

Your project is awesome.

Demand the best audio.

music • sound • voice

the sound department

415.454.8988

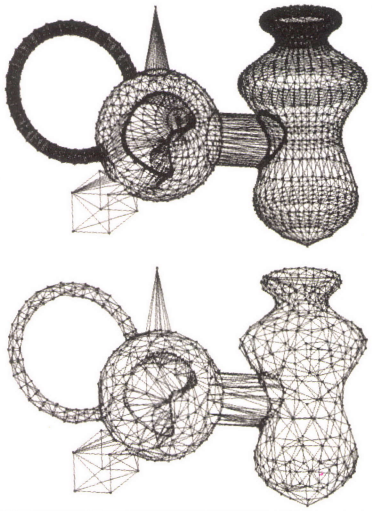
www.thesounddepartment.com

POPLib 2.0

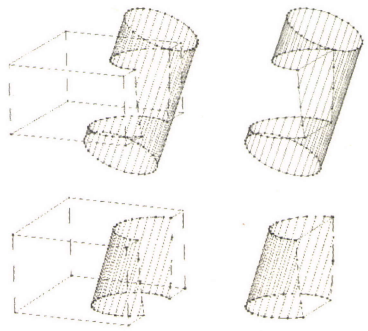
Polygon Optimization Library

POPLib 2.0 is available as C++ Source
Includes: Tessellation; Decimation; Booleans; Sectioning; Ray Firing; and other Polygon and Geometry Tools
Great tools for Software Developers!

Decimation



Booleans



www.PolygonLibrary.com
Click Here for Fewer Polygons!
Click Here for Better Polygons!
Click Here for Faster Polygons!

+1.858.538.3800

info@PolygonLibrary.com

continued from page 64

obliged to hand over the controls to the other people waiting to play, so there may be other aspects to *Sam* that I did not fully explore. The museum's web site had a short interview with the artist, and, when asked about the violent nature of the piece, the artist responded that the life of Sam is directly related to anger and frustration. He said the idea of destruction can be simultaneously pleasant and painful, and the aggressive aspects of play cannot be isolated from the reality in which we live. Another interesting note is that *Sam* was shown in a section of the museum called "Les Tokyo Games," which is set aside strictly for the artistic exploration of the medium of videogames.

The acceptance of these recent works is an extension of the art world's embrace of digital media over the past two years (marked in my mind by the simultaneous technology shows at the Whitney and the San Francisco Museum of Modern Art last year) although, in the case of *Sam*, I'd wager there's also a tie to the growing, government-supported French game industry. With the recent boom in digital art, the museums are more than open to the medium of digital interactive media, and in some cases they may even still be partially biased to the potential of such an exciting new medium.

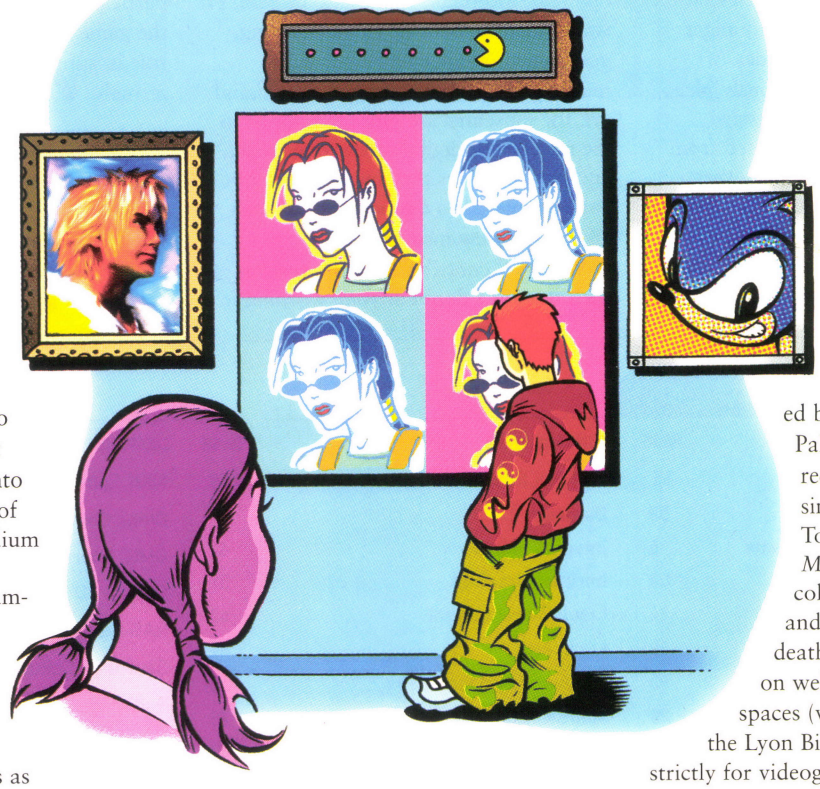
Who will contribute to the creation of such works? I doubt anyone working in a commercial medium that rakes in 9 billion smackers a year is ready to cut off an ear just yet. But it is an exciting prospect to work in a medium with the potential for such artistic expression, and I hope that, just as the commercial film industry has learned and grown from participation in the indie/art film scene, the game industry also will take an active role in supporting its own cultural growth.

PAUL WARNE | *Paul is a video/installation artist residing in San Francisco, working most recently as a level designer for Double Fine Productions. He would like to thank D.H. Penn for his assistance. Comments are welcome at paul@dept-h.com.*

ADVERTISER INDEX

COMPANY NAME	PAGE	COMPANY NAME	PAGE	COMPANY NAME	PAGE
3DLabs	15.	Full Sail Real World Education	59	NXN Software	53
Academy of Art College	58	Gamespy	21	Okino Computer Graphics	14
Analog Devices/Staccato Systems	5	Integrity Ware	62	OMNI Interactive Video	61
Apple Computer	C2	Intrinsic Graphics	51	Oregon 3D	13
Charles River Media	14	Liquid Development	62	RAD Game Tools	C4
Cogswell Polytechnical	60	LucasArts	57	Savannah College of Art & Design	60
Conitec	63	Mesmer Animation Labs	60	Softimage	C3
Cyberware	23	Metrowerks	25	The Sound Department	62
Discreet	9	Microsoft Corporation	17	University of Advanced Computer Technology	59
Eaton	7	Morgan Kaufmann Publishers	27		
Eyematic	57	New Pencil	61		
From Art to Design Inc.	61	Numerical Design Ltd.	2	Vancouver Film School	59

The Games of Art



Angst-filled cries can be heard from time to time within the development community that videogames receive little or no recognition from the art world. Without going into whether most products of this industry-fueled medium merit such recognition, there are two recent examples worthy of mention that have demonstrated the willingness of art museums to accept pieces created from digital interactive games as "higher" art.

The first is a video piece by well-known New York avant-garde film artist Peggy Ahwesh, titled *She-Puppet*. Though not an interactive piece, *She-Puppet* is worth mentioning for its use of the game TOMB RAIDER as its focus, and because of the recognition it has received on the festival circuit, most notably with its inclusion in this year's Whitney Biennial.

She-Puppet is a 15-minute video collage of captured footage of the artist "playing" the game. The video is overdubbed with quotes from philosophical texts and from such eclectic sources as Portuguese author Fernando Pessoa, science-fiction writer Joanna Russ, and interstellar jazz musician Sun Ra. I say "playing," as it is really the artist acting as puppeteer of Lara Croft, placing her in situations that have nothing to do with achieving the goals of the game.

Ahwesh guides Lara on an existential romp, questioning her own existence as a videogame character (much like a low-tech, highbrow mix of *The Onion*'s article "Video-Game Character Wondering Why Heartless God Always Chooses 'Continue'" and the QUAKE Machinimas). Ahwesh allows a swarm of vultures to slowly kill Lara, and, in another sequence, the artist edits together a looped series of Lara's death animations. When asked about these death sequences Ahwesh said, "The death throes are the most deeply pleasurable [part of the game] . . . It's like a sex act."

In contrast to *She-Puppet* is a piece titled *Sam*, which I found in Paris's Site de Creation Contemporaine, a museum housed in the Palais de Tokyo. Where *She-Puppet* is a video piece whose content is based on the medium of videogames, *Sam* is a fully interactive HALF-LIFE mod created by Stockholm native Palle Torsson, who received recognition for a similar piece he did with Tobias Bernstrup titled *Museum Meltdown*, a collection of HALF-LIFE and DUKE NUKEM 3D deathmatch levels modeled on well-known museum spaces (which was featured in the Lyon Biennial in a section strictly for videogames).

Sam is projected onto a large movie screen located in a small, dark room in the middle of the museum space. Navigating with a mouse and joystick, you interact with the piece by taking on the role of a young girl named Sam. You begin in Sam's happy little girl's room in her typical suburban home. But it just so happens that this little girl is packing heat. After shooting up her room, you proceed downstairs and execute her parents. You then help Sam escape to the outside world, where you are free to massacre shopkeepers and pedestrians in a small town area. You then go to her school, where the violence continues. This, at least, was how I chose to interact with the art.

It was quite a popular piece, and I felt

continued on page 63

Illustration by Dominic Bugatto

"[REAL-TIME SHADERS] ALLOW FOR PREVIEWING OF EFFECTS ON THE XSI VIEWPORT **DURING CONTENT CREATION**, SOMETHING THAT HAS PREVIOUSLY ONLY BEEN POSSIBLE ON **THE XBOX**."
– Yuichi Masuya, Microsoft Co., Ltd., Xbox Division

WHAT YOU SEE IS WHAT YOU GET. ONLY IN **XSI**™ v.**2.0**

Artists and developers, here's the game plan: Start with **SOFTIMAGE|XSI** v.**2.0**, the only true creative choice for sophisticated real-time effects.

XSI Real-Time Shaders deliver **state-of-the-art shading** control along with the industry's only native **nonlinear animation (NLA)** system for action-based motion. Now you can work with your art the same way your game engine does.

Feel like calling the shots yourself? XSI v.2.0 internals are fully exposed with the new integrated **Software Development Kit (SDK)** and file I/O is seamless via the free, extensible **dotXSI™ file format**.

Ready to play? Visit **www.softimage.com/v2**, order the new XSI Experience CD and **let the games begin**.

2.0

SOFTIMAGE® | XSI™

Innovate. Create. Collaborate.

AMD Athlon MP
AMD Certified

© 2002 Avid Technology, Inc. All rights reserved. Softimage and Avid are registered trademarks and XSI, the XSI Logo, dotXSI and Make, Manage and Move Media are trademarks of Avid Technology, Inc. AMD, the AMD logo, AMD Athlon MP and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other trademarks contained herein are the property of their respective owners.

Avid
Make, Manage and Move Media™

The Best in Game Development Technology

Granny 3D 2.0

Granny 2.0 combines the **sophisticated dynamic 3D animation system** of its predecessor with a completely new and optimized run-time engine to deliver incredible performance in an extraordinarily small memory footprint perfect for consoles as well as PCs and Macs. Brand new, fully customizable and scriptable **MAX & Maya** plug-ins export all material and shader graph data, multiple texture and color channels for vertices, complete skinning information, and user-defined node attributes and scripts, all to a new, completely open, extensible, and easy-to-use file format.

With all this, plus utilities for advanced camera control, collision detection, animation and texture processing, MIP-map generation, and **Bink and S3TC** texture compression, Granny adds years worth of features to your 3D engine in just a few hours of integration time!

Bink Video Technology

The very best video codec - make your videos shine! Unrivaled quality (**better than DVD** and MPEG II). Up to **three times faster** than other true-color codecs. Perceptibly lossless, **8 to 1 audio compression**. Decompresses to DirectDraw, DIBSections, YUV overlays, or any other memory. Available for Win32, MacOS, Xbox, and now **Nintendo GameCube**. Think Bink!

Smacker Video Technology

The best 256-color codec on the planet! Smacker is the codec of choice for **256-color games** (of course), games targeting Pentium 133s and below, games with video sprites or video transparency (**much faster in 256 colors**), or games with extremely high-resolution videos (800x600 and up). Smacker is available on Win32, Win16, DOS, and MacOS.

Miles Sound System

The ultimate sound system for PC and Mac! Miles supports 2D and 3D audio, MIDI with DLS, streaming, CD audio, **DSP filtering**, **MP3 playback (patent rights included)**, **Internet voice chat** (2900, 2400, and 1200 bps codecs), Creative **EAX 1, 2 & 3**, Aureal **A3D 1 & 2**, RSX 3D, DirectX 7, Dolby Surround, QSound, **super-fast software EAX emulation**, and more!

RAD GAME TOOLS ©

425 893 4300
www.radgametools.com

POWERFUL TECHNOLOGY. EASY TO USE. NO ROYALTIES.